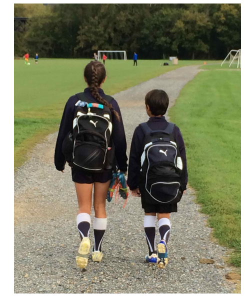
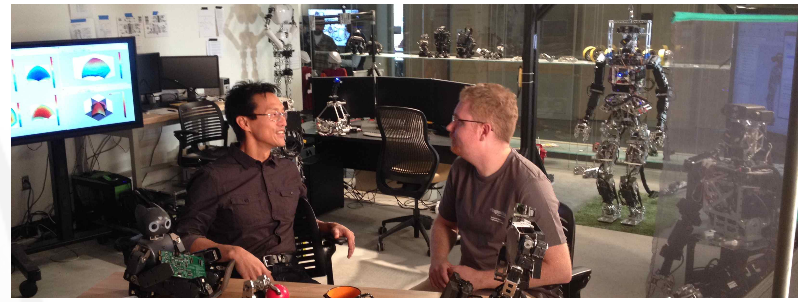
An Ecosystem for the New HPC: Heterogeneous Parallel Computing

Wu FENG

Director
SEEC Center (Synergistic Environments for Experimental Computing)
Professor and Elizabeth & James E. Turner Fellow
Dept. of Computer Science
Dept. of Electrical & Computer Engineering
Health Sciences
Virginia Bioinformatics Institute

A Little Bit About Me …

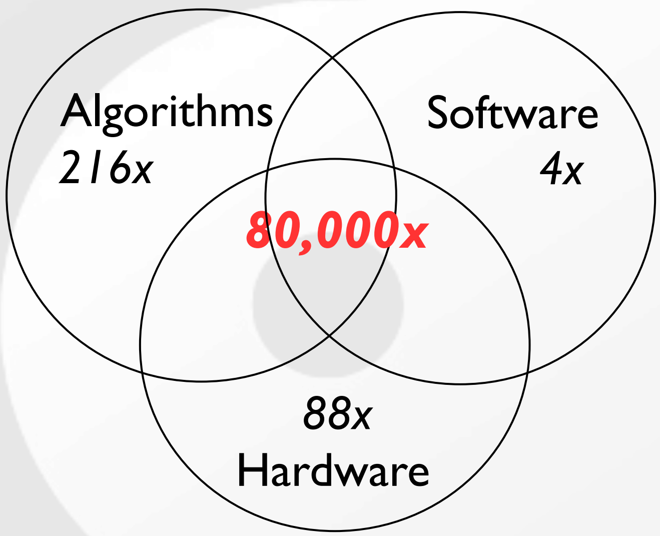



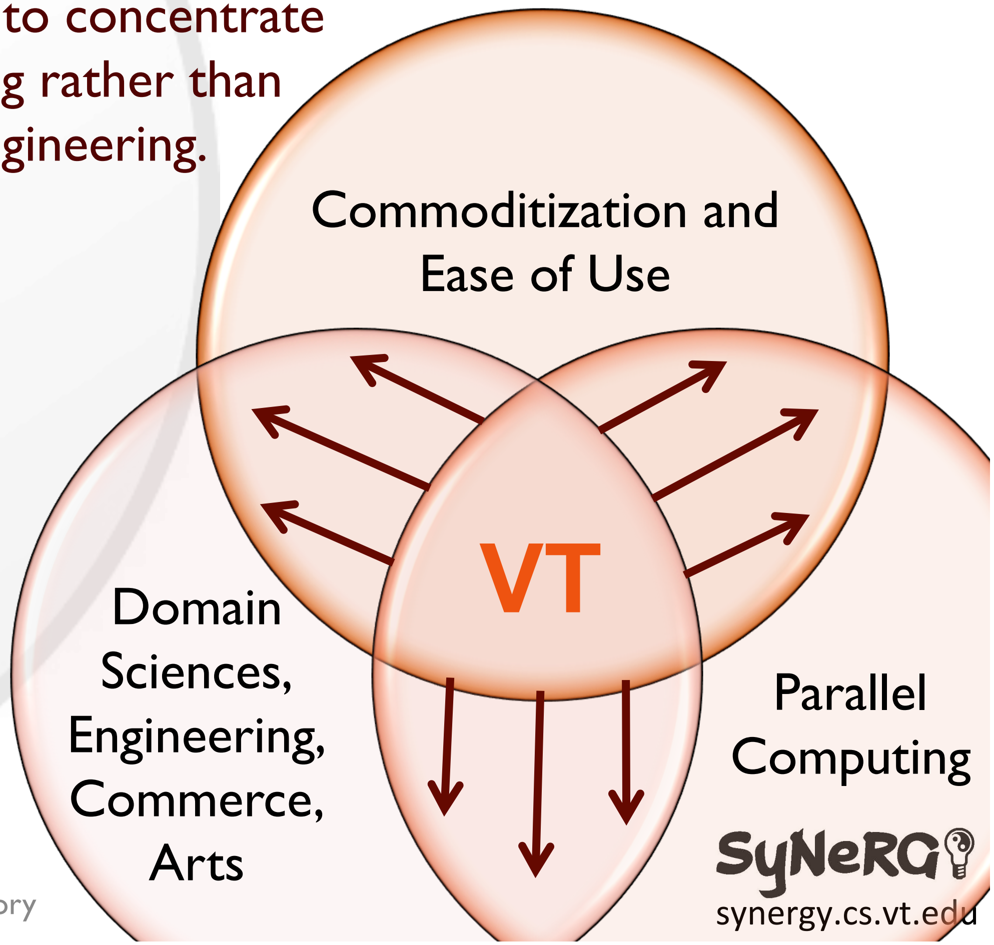


- Education
  - Ph.D., Computer Science
    U. Illinois at Urbana-Champaign, 1996
- Professional
  - Current Appointments
    - Professor and Elizabeth & James Turner Fellow; Departments of Computer Science, Electrical & Computer Engineering, and Health Sciences; Virginia Tech
    - Director, SyNeRGy Laboratory (http://synergy.cs.vt.edu/) → SEEC Center
    - Founder, The Green500 (http://www.green500.org/)
    - Adjunct Faculty, Virginia Bioinformatics Institute, Virginia Tech
  - Previous Appointments & Professional Stints
    - *Academia:* The Ohio State U. ('00-'03), Purdue U. ('98-'00), U. of Illinois at Urbana-Champaign ('96-'98).
    - *Government:* Los Alamos Nat'l Lab ('98-'06), NASA Ames Research Ctr ('93)
    - *Industry:* IBM T.J. Watson Rsch ('90), Vosaic ('97), Orion Multisystems ('04-'05)

A Little Bit About My Research …

- *Basic and applied research* in efficient parallel and distributed computing (in the small and the large) via the synergistic co-design of hardware, software, and algorithms
  - Enable scientists and engineers to concentrate on their science and engineering rather than on the *computer* science and engineering.





http://www.youtube.com/watch?v=zPBFenYg2Zk

An Ecosystem for the New HPC: Heterogeneous Parallel Computing

Wu FENG

Professor and Elizabeth & James E. Turner Fellow

Dept. of Computer Science

Dept. of Electrical & Computer Engineering

Health Sciences

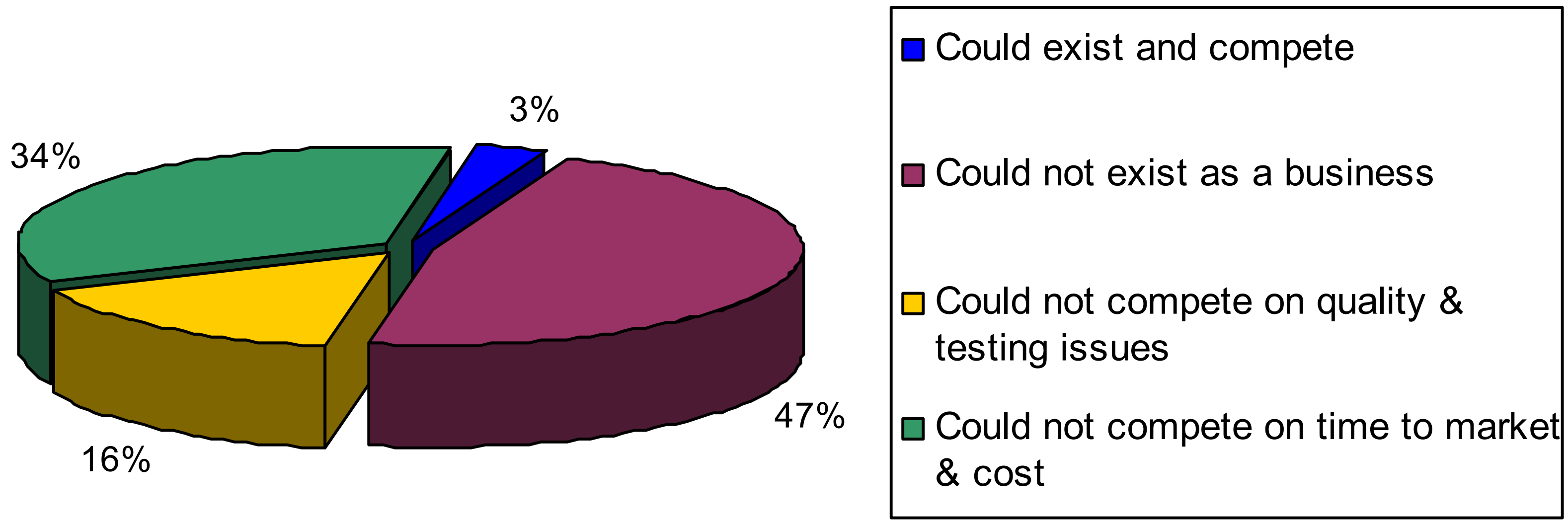
Virginia Bioinformatics Institute

Japanese ‘Computnik’ Earth Simulator Shatters U.S. Supercomputer Hegemony

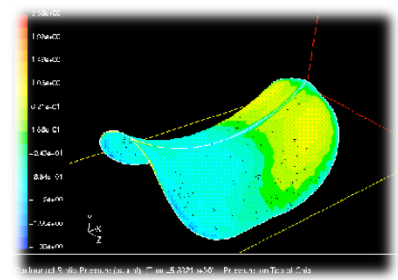
Tokyo 20 April 2002 *The Japanese Earth Simulator is on-line and producing results that alarm the USA, that considered itself as being leading in supercomputing technology. With over 35 Tflop/s, it five times outperforms the Asci White supercomputer that is leading the current TOP500 list. No doubt that position is for the Earth Simulator, not only for the next list, but probably even for*

Importance of High-Performance Computing (HPC)

Competitive Risk From Not Having Access to HPC



Data from Council of Competitiveness.
Sponsored Survey Conducted by IDC



Only 3% of companies could exist and compete *without* HPC.

- 200+ participating companies, including many Fortune 500 (Proctor & Gamble and biological and chemical companies)

VirginiaTech
Invent the Future

Los Alamos National Laboratory

SyNeRGy
synergy.cs.vt.edu

The New York Times

Business Day

Technology

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH

Search Technology

Go

Inside Technology

Internet | Start-Ups | Business Computing

China Wrests Supercomputer Title From U.S.

By ASHLEE VANCE

Published: October 28, 2010

A Chinese scientific research center has built the fastest supercomputer ever made, replacing the United States as maker of the swiftest machine,

RECOMMEND

TWITTER

LINKEDIN

Computnik 2.0?

- The Second Coming of Computnik? Computnik 2.0?
  - No … "only" 43% faster than the previous #1 supercomputer, *but*
    - → $20M cheaper than the previous #1 supercomputer
    - → 42% less power consumption
- The Second Coming of the "Beowulf Cluster" for HPC
  - The further commoditization of HPC

The First Coming of the "Beowulf Cluster"

- Utilize *commodity* PCs (with commodity CPUs) to build a supercomputer

The Second Coming of the "Beowulf Cluster"

- Utilize *commodity* PCs (with *commodity* CPUs) to build a supercomputer

+

- Utilize *commodity* graphics processing units (GPUs) to build a supercomputer

Issue: Extracting *performance* with *programming ease* and *portability* → *productivity*

In other words, can we achieve "Performance + Programmability + Portability = Productivity?"

CPU Computing
(General Purpose)

+

GPU Computing
(Specialized)

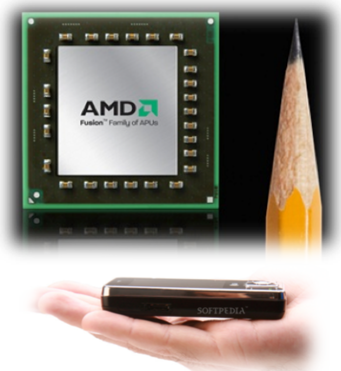
CPU + GPU ≈ Left Brain + Right Brain

Heterogeneous Parallel Computing

"Holy Grail" Vision

- Ecosystem for the New HPC: Heterogeneous Parallel Computing

#96 on TOP500 (11/11)

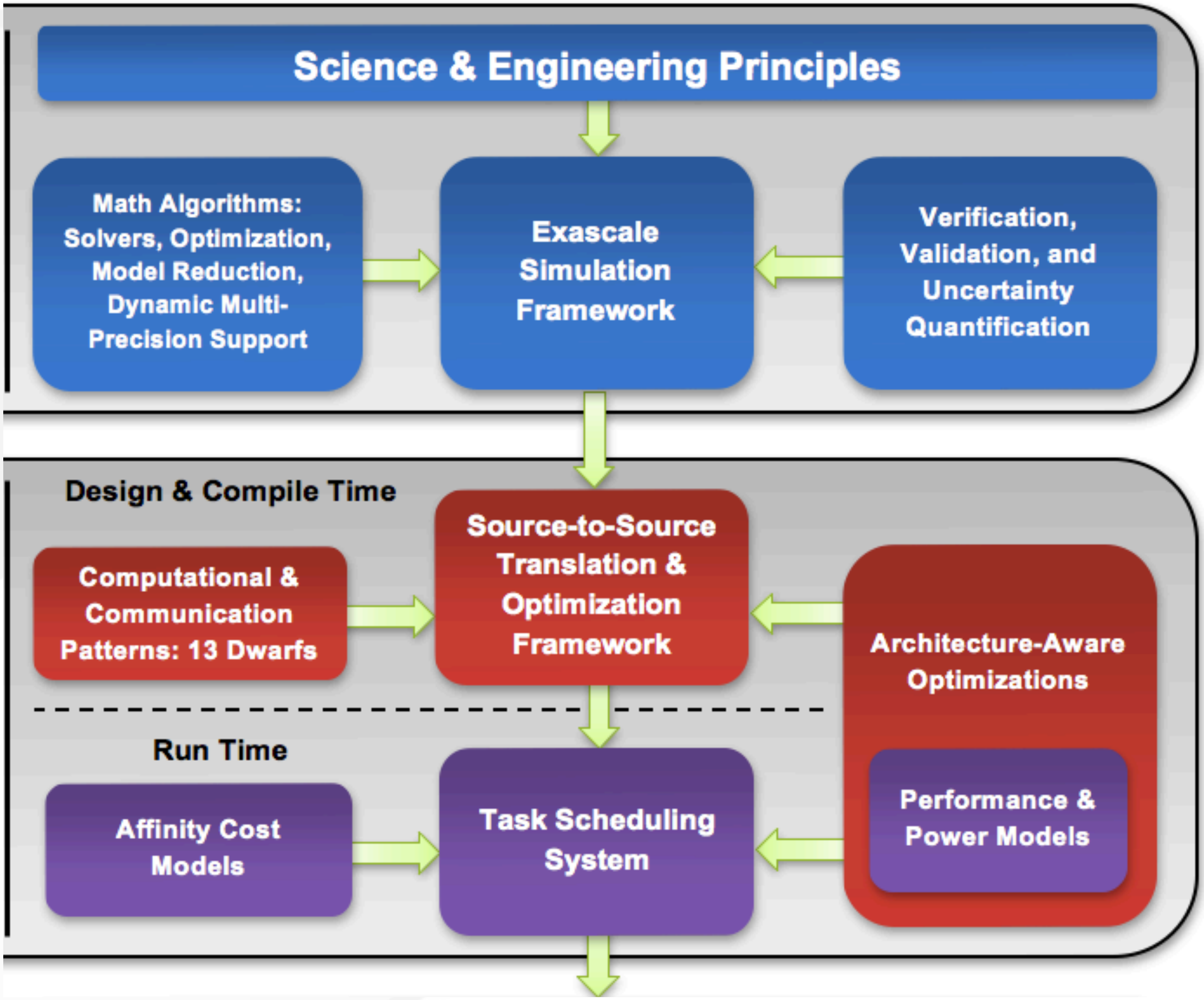


Highest-ranked commodity supercomputer in U.S. on the Green500 (11/11)

  – Enabling software that tunes parameters of hardware devices
    *… with respect to **performance, programmability, and portability***
    … via a benchmark suite of dwarfs (i.e., motifs) and mini-apps

Design of Composite Structures

Software Ecosystem



Heterogeneous Parallel Computing (HPC) Platform

VirginiaTech
Invent the Future

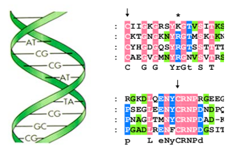
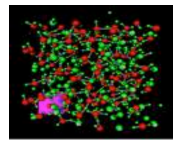
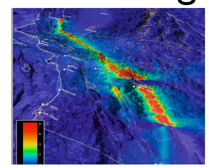
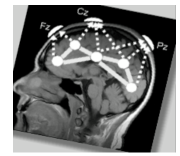
SyNeRG
synergy.cs.vt.edu

Intra-Node

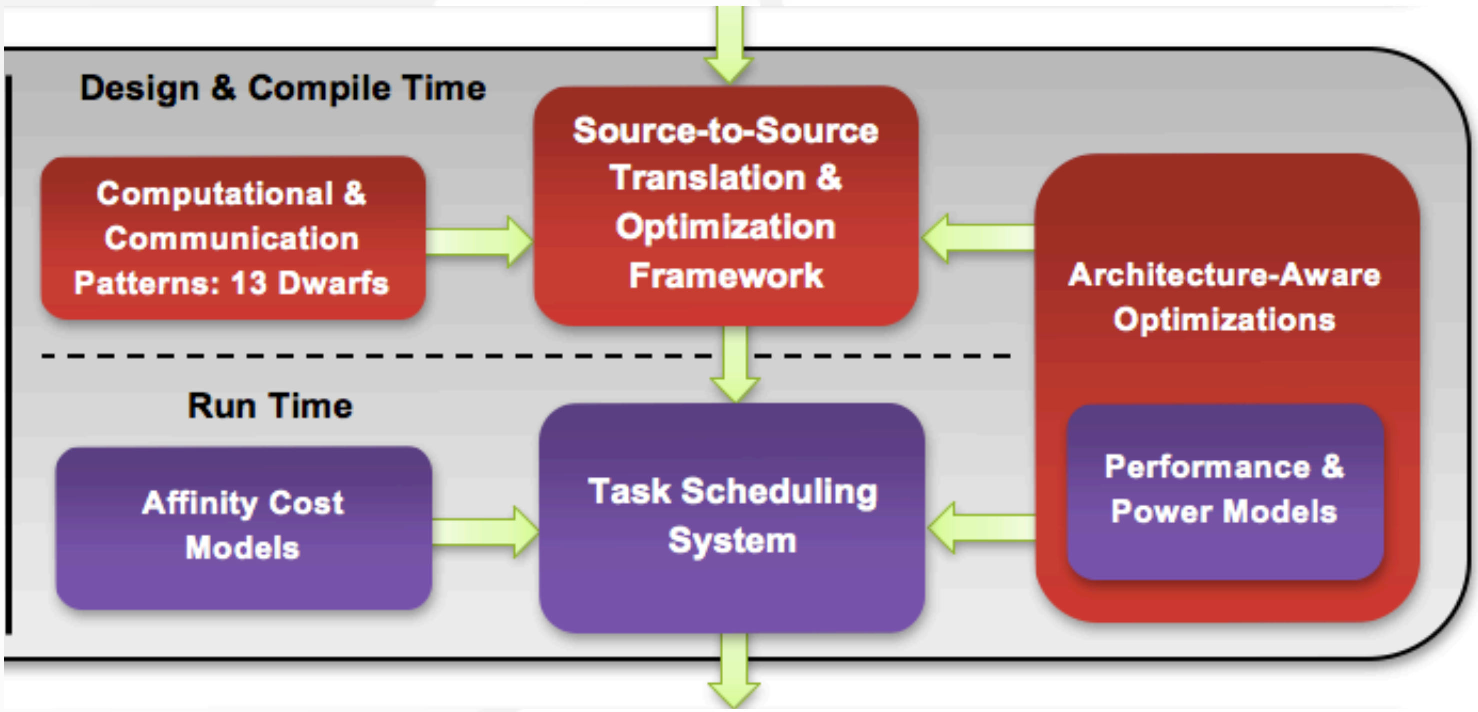
An Ecosystem for Heterogeneous Parallel Computing

^

Applications

Sequence Alignment	Molecular Dynamics	Earthquake Modeling	Neuro-informatics	CFD for Mini-Drones
				

Software Ecosystem



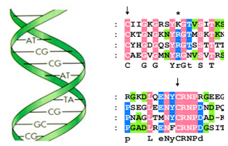
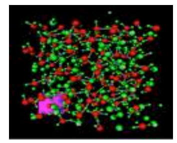
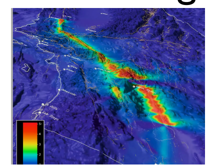
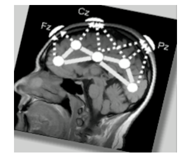
Heterogeneous Parallel Computing (HPC) Platform

VirginiaTech *Invent the Future*

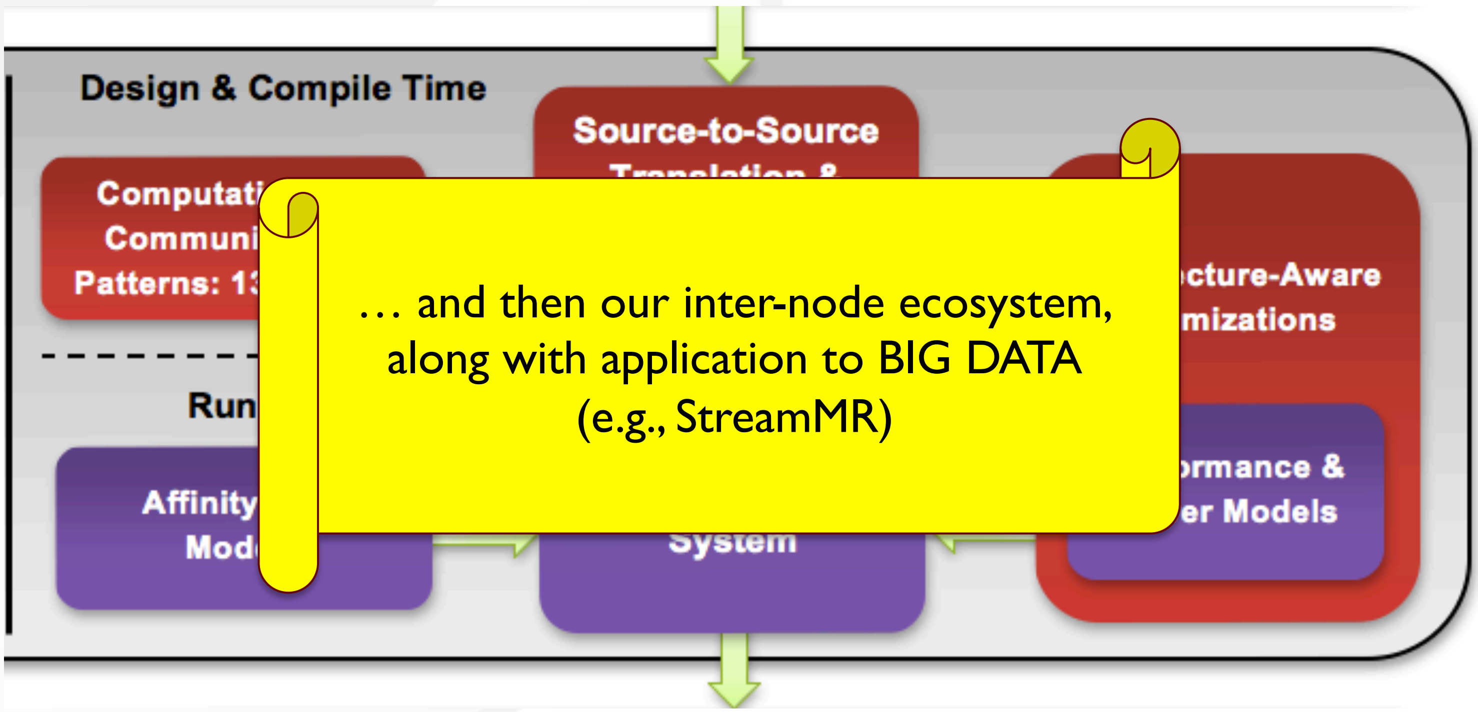
SyNeRG
synergy.cs.vt.edu

Intra-Node
An ^ Ecosystem for Heterogeneous Parallel Computing

Applications

Sequence Alignment	Molecular Dynamics	Earthquake Modeling	Neuro-informatics	CFD for Mini-Drones
				

Software Ecosystem

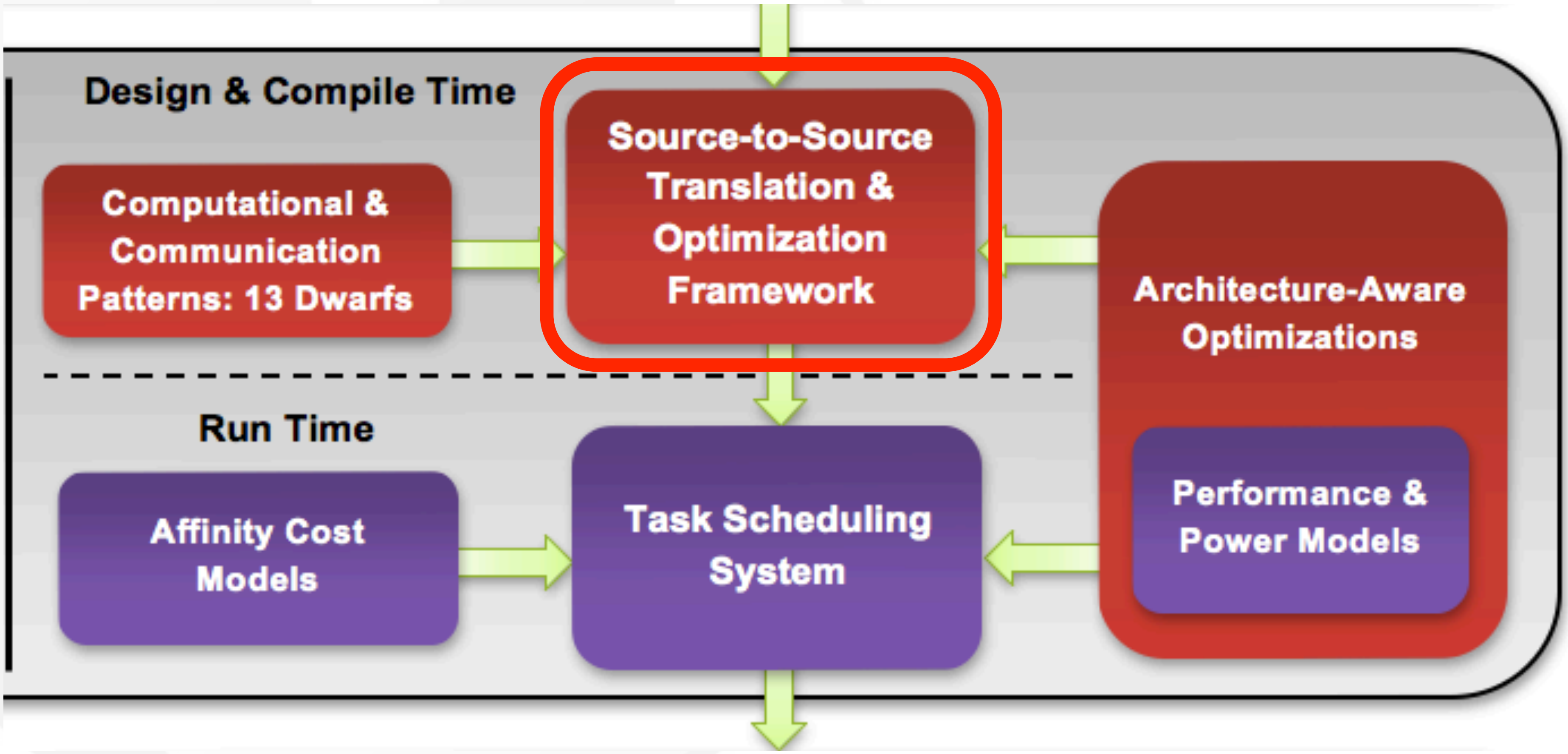


… and then our inter-node ecosystem, along with application to BIG DATA (e.g., StreamMR)

Heterogeneous Parallel Computing (HPC) Platform

Roadmap

Goal: Minimize the re-writing of code, e.g., CFD for mini-drones.
CUDA → OpenCL and OpenMP → OpenACC



Programming GPUs

CUDA

- NVIDIA's proprietary framework

OpenCL

- Open standard for heterogeneous parallel computing (Khronos Group)
- Vendor-neutral environment for CPUs, GPUs, APUs, and even FPGAs

OpenACC

- An extension of OpenMP for GPUs

CUDA-Accelerated Applications

GOVERNMENT & DEFENSE
Ikena: Imagery Analysis and Video Forensics
Signal Processing Library: GPU VSIPL
IDL and MATLAB® Acceleration: GPULib
GIS: Manifold

MOLECULAR DYNAMICS, COMPUTATIONAL CHEMISTRY
OpenMM library for molecular dynamics on GPUs
GROMACS using OpenMM
NAMD molecular dynamics
VMD visualization of molecular dynamics
HOOMD molecular dynamics
Acellera: ACEMD bio-molecular dynamics package
BigDFT: DFT (Density functional theory) electronic structure
MDGPU
GPUGrid.net

LIFE SCIENCES, BIO-INFORMATICS
GPU HMMER
DNA Sequence alignment: MUMmerGPU
LISSOM: model of human neocortex using CUDA
Silicon Informatics: AutoDock

ELECTRODYNAMICS AND ELECTROMAGNETIC
Acceleware: FDTD Solver
Acceleware: EM Solutions
Remcom XStream FDTD
SPEAG Semcad X
CST Microwave Studio
Quantum electrodynamics library
GPMAD : Particle beam dynamics simulator

MEDICAL IMAGING, CT, MRI
RealityServer
GPULib:IDL acceleration
Acceleware: Imaging Solutions
Digisens: SnapCT tomographic reconstruction software
Techniscan: Whole Breast Ultrasound Imaging System

OIL & GAS
Acceleware: Kirchoff and Reverse Time Migration
SeismicCity: 3D seismic imaging for prestack depth migration
OpenGeoSolutions: Spectral decomposition and inversion
Mercury Computer systems: 3D data visualization
ffA: 3D Seismic processing software
Headwave: Prestack data processing

FINANCIAL COMPUTING AND OPTIONS PRICING
SciComp: derivatives pricing
Hanweck: options pricing
Exegy: Risk Analysis
Aqumin: 3D Visualization of market data
Level 3 Finance
OnEye (Australia): Accelerated Trading Solutions
Arbitragis Trading

MATLAB, LABVIEW, MATHEMATICA, R
CUDA Acceleration for MATLAB
Accelereyes: Jacket™ engine for MATLAB
GPULib: mathematical functions for IDL and MATLAB
Integrating Simulink with CUDA using S-functions
Enabling GPU Computing in the R Statistical Environment
Mathematica plug-in for CUDA
National Instruments LabView for NVIDIA GPUs

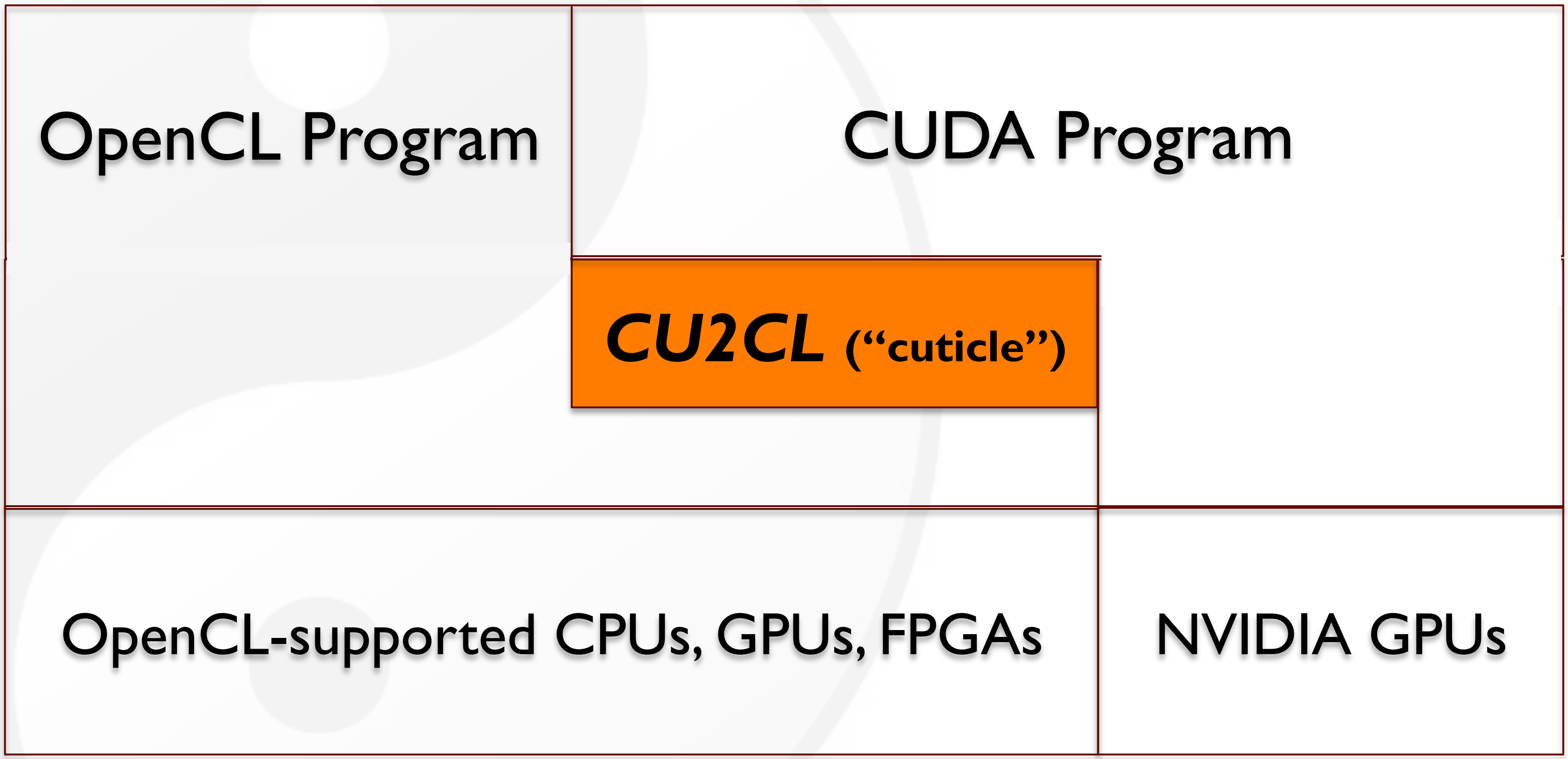
ELECTRONIC DESIGN AUTOMATION
Agilent EESof: ADS SPICE simulator
Synopsys: Sentaraus TCAD
Gauda: Optical proximity correction (OPC)

WEATHER AND OCEAN MODELING
CUDA-accelerated WRF code

VIDEO, IMAGING, AND VISION APPLICATIONS
Axxon Intellect Enterprise Video Surveillance Software
Pflow CUDA Plugin for Autodesk 3ds Max
RUINS Shatter CUDA Plug-in for Maya
Bullet 3D Multi-Physics Library with CUDA Support
CUDA Voxel Rendering Engine
Furryball: Direct3D GPU Rendering Plugin for Maya

See: http://www.nvidia.com/object/cuda_app_tesla.html

OpenCL: Write Once, Run Anywhere



CU2CL: CUDA-to-OpenCL Source-to-Source Translator†

- Works as a Clang plug-in to leverage its production-quality compiler framework
  - Update: Works as a standalone app!
- Covers primary CUDA constructs found in CUDA C and CUDA run-time API.
- Delivers performance portability when OpenCL 1.2-equivalent CUDA code run on same NVIDIA GPU.
- Focuses on *functional portability* … for now.

† "CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-core Architectures," *17th IEEE Int'l Conf. on Parallel & Distributed Systems (ICPADS)*, Dec. 2011.

Why CU2CL?

- Much larger # of apps implemented in CUDA than in OpenCL
  - Idea
    - Leverage scientists' investment in CUDA to drive OpenCL adoption
  - Issues (from the perspective of *domain* scientists)
    - Writing from Scratch: Learning Curve
      (OpenCL is too low-level an API compared to CUDA. CUDA also low level.)
    - Porting from CUDA: Tedious and Error-Prone
- Significant demand from major stakeholders (and Apple …)

Why *Not* CU2CL?

- Just start with OpenCL?!
- CU2CL only does *source-to-source translation* at present
  - **Functional (code) portability? Yes.**
  - **Performance portability? Mostly no.**

Programmability & Portability vs. Performance

C

```
void MatMul(float* M, float* N,
            float* P, int W) {
  for (int i=0;i<W;++i)
    for (int j=0;j<W;++j) {
      for (int k=0;k<W;++k) {
        P[i*W+j] += M[i*W+k]*
                    N[k*W+j];
      }
    }
}
```

OpenACC

```
void MatMul(float * restrict M,
   float * restrict N,
   float* restrict P,  int W) {
  int i, j, k ;
  #pragma acc kernels
  copyout(P[0:(W*W)]),
  copyin(M[0:  (W*W)],N[0:(W*W)])
  for (i=0; i<W; i++){
    for (j=0; j<W; j++) {
      for (k=0; k<W; k++)
        P[i*W+j]+=M[i*W+k]*
                   N[k*W+j] ;
      }
    }
}
```

CUDA

```
float *d_M, *d_N, *d_P;
int matrix_size=Width*Width*sizeof(float);
cudaMalloc(&d_M, matrix_size);
cudaMemcpy(d_M, M, matrix_size,
           cudaMemcpyHostToDevice);
cudaMalloc(&d_N, matrix_size);
cudaMemcpy(d_N, N, matrix_size,
           cudaMemcpyHostToDevice);
cudaMalloc(&d_P, matrix_size);
dim3 dimGrid(1,1);
dim3 dimBlock(Width,Width);
MatMul<<<dimGrid, dimBlock>>>(d_M, d_
N, d_P,Width);
cudaMemcpy(P,d_P,matrix_size,
           cudaMemcpyDeviceToHost);
cudaFree(d_P);
cudaFree(d_M);
cudaFree(d_N);
```

```
__global__
void MatMul(float* d_M,
            float* d_N,
            float* d_P,
            int W) {
  int row = threadIdx.y;
  int col = threadIdx.x;
  float P_val = 0;
  for (int k = 0; k < W; ++k) {
    float M = d_M[row * W+ k];
    float N = d_N[k * W+ col];
    P_val += M*N;
  }
  d_p[row*W+col] = P_val;
}
```

OpenCL

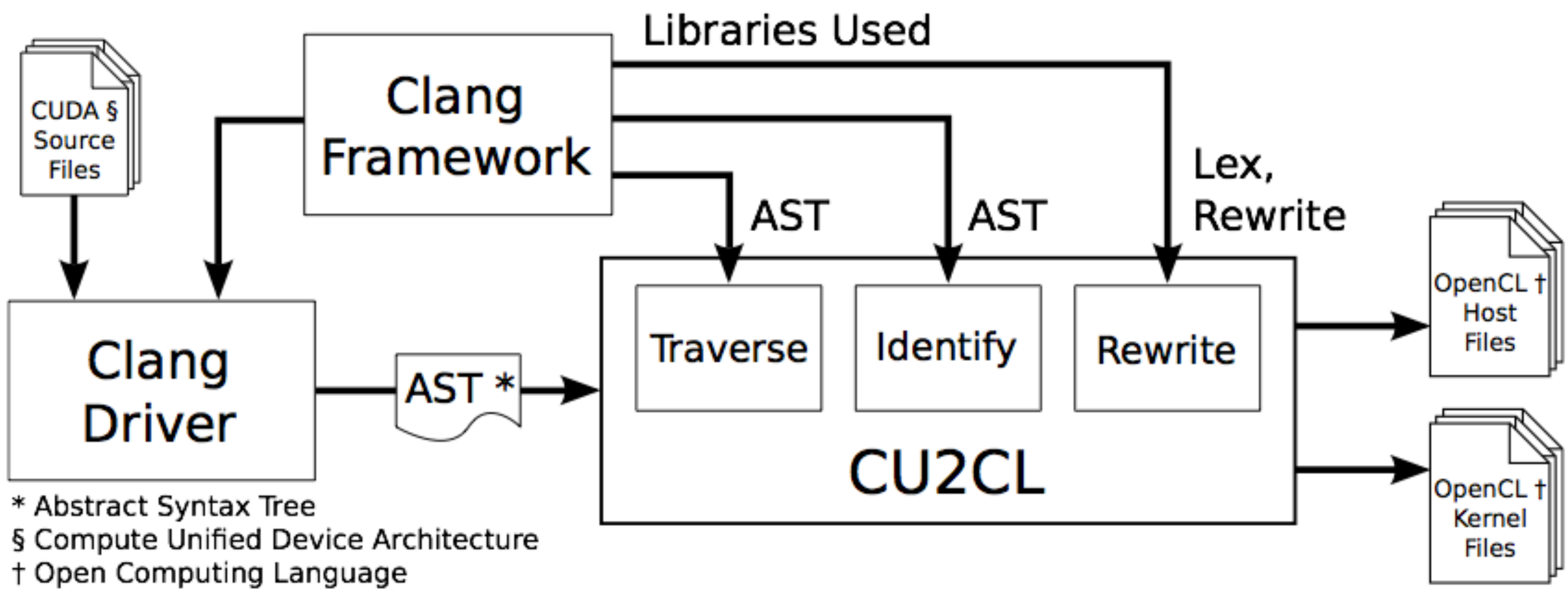
```
/*Code contains parts adapted from code originally written by Tim Mattson
and obtained from: https://github.com/HandsOnOpenCL/Exercises-Solutions/blob/master/Solutions/Exercise08/
*/
char * kernelsource;
cl_int err;
cl_device_id device;
cl_context context;
cl_command_queue commands;
cl_program program;
cl_kernel kernel;
size = W*W;
h_A = (float *)malloc(size*sizeof(float));
h_B = (float *)malloc(size*sizeof(float));
h_C = (float *)malloc(size*sizeof(float));
cl_uint deviceIndex = 0;
parseArguments(argc, argv, &deviceIndex);
cl_device_id devices[MAX_DEVICES];
unsigned numDevices = getDeviceList(devices);
if (deviceIndex >= numDevices)
{
  printf("Invalid device index\n");
  return EXIT_FAILURE;
}
device = devices[deviceIndex];
char name[MAX_INFO_STRING];
getDeviceName(device, name);
printf("\nUsing OpenCL device: %s\n", name);
context = clCreateContext(0, 1, &device, NULL, NULL, &err);
checkError(err, "Creating context");
commands = clCreateCommandQueue(context, device, 0, &err);
checkError(err, "Creating command queue");
d_a = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                     sizeof(float) * size, h_A, &err);
checkError(err, "Creating buffer d_a");
d_b = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                     sizeof(float) * size, h_B, &err);
checkError(err, "Creating buffer d_b");
d_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
                     sizeof(float) * size, NULL, &err);
checkError(err, "Creating buffer d_c");
kernelsource = getKernelSource("matmul.cl");
program = clCreateProgramWithSource(context, 1, (const char **) &kernelsource, NULL, &err);
checkError(err, "Creating program with matmul.cl");
free(kernelsource);
err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
if (err != CL_SUCCESS)
{
  size_t len;
  char buffer[2048];
  printf("Error: Failed to build program executable!\n%s\n", err_code(err));
  clGetProgramBuildInfo(program, device, CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer, &len);
  printf("%s\n", buffer);
  return EXIT_FAILURE;
}
kernel = clCreateKernel(program, "MatMul", &err);
checkError(err, "Creating kernel with matmul.cl");

err =  clSetKernelArg(kernel, 0, sizeof(cl_mem),    &d_a);
    err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_b);
    err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &d_c);
    err |= clSetKernelArg(kernel, 3, sizeof(int), &W);

checkError(err, "Setting kernel args");
const size_t global[2] = {W,W};
err = clEnqueueNDRangeKernel(
        commands,
        kernel,
        2, NULL,
        global, NULL,
        0, NULL, NULL);
checkError(err, "Enqueueing kernel");
err = clFinish(commands);
checkError(err, "Waiting for kernel to finish");
err = clEnqueueReadBuffer(
        commands, d_c, CL_TRUE, 0,
        sizeof(float) * size, h_C,
        0, NULL, NULL);
checkError(err, "Reading back d_c");
```

```
__kernel
void MatMul( global float* M,
             global float* N,
             global float* P,
             int W) {
 int tx=get_global_id(0);
 int ty=get_global_id(1);
 for(int k=0; k<W; ++k) {
  value+=A[ty*W+k]*B[k*W+tx];
  }
  C[ty*W+tx]=value;
}
```

… but portable!

Most Programmable → **Least Programmable**

CU2CL Translation and Performance



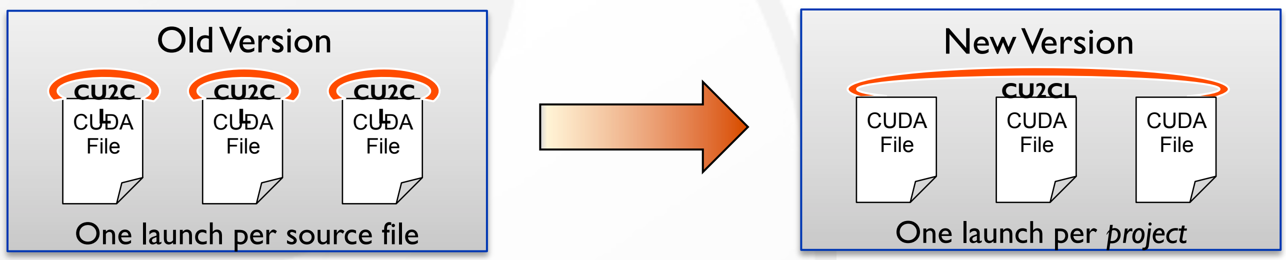
- Automatically translated OpenCL codes (via CU2CL) yield similar execution times to manually translated OpenCL codes (when running on the same device)

	Application	CUDA Lines	Lines Manually Changed	% Auto-Translated
CUDA SDK	bandwidthTest	891	5	99
CUDA SDK	BlackScholes	347	14	96
CUDA SDK	matrixMul	351	9	97
CUDA SDK	vectorAdd	147	0	100
Rodinia	Back Propagation	313	24	92
Rodinia	Hotspot	328	2	99
Rodinia	Needleman-Wunsch	430	3	99
Rodinia	SRAD	541	0	100
Delaware	Fen Zi: Molecular Dynamics	17,768	1,796	90
VT	GEM: Molecular Modeling	524	15	97
AMD	IZ PS: Neural Network	8,402	166	98

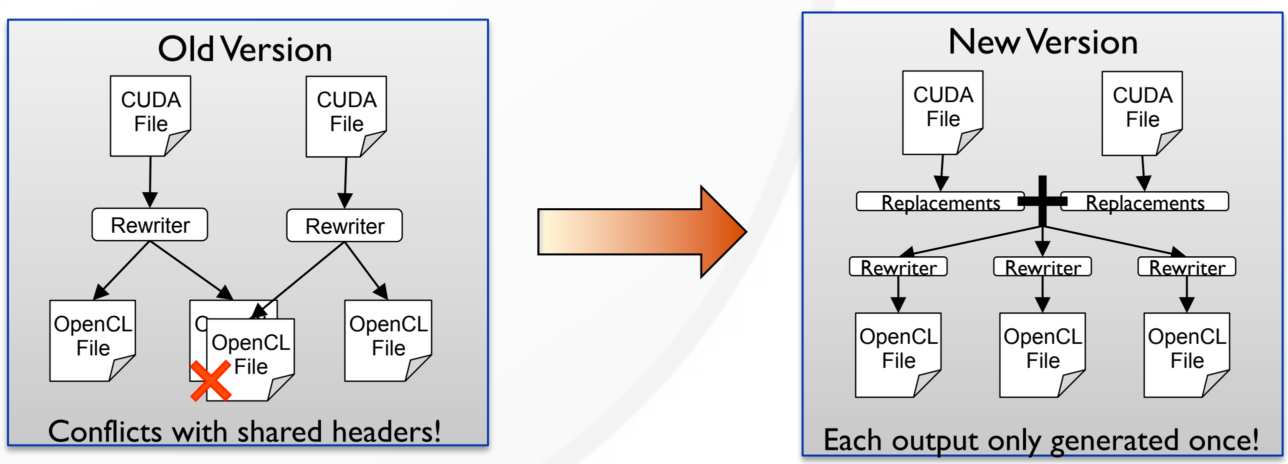
Evaluating & Enhancing Productivity with CU2CL

See Github and http://chrec.cs.vt.edu/

- Binary: CU2CL v0.7.0b (May 2015)
  - Major revision – Multi-file Translation Support
  - Overhauled invocation – "One-shot" translation of all files



  - Overhauled OpenCL generation – Intelligent merging and de-duplication

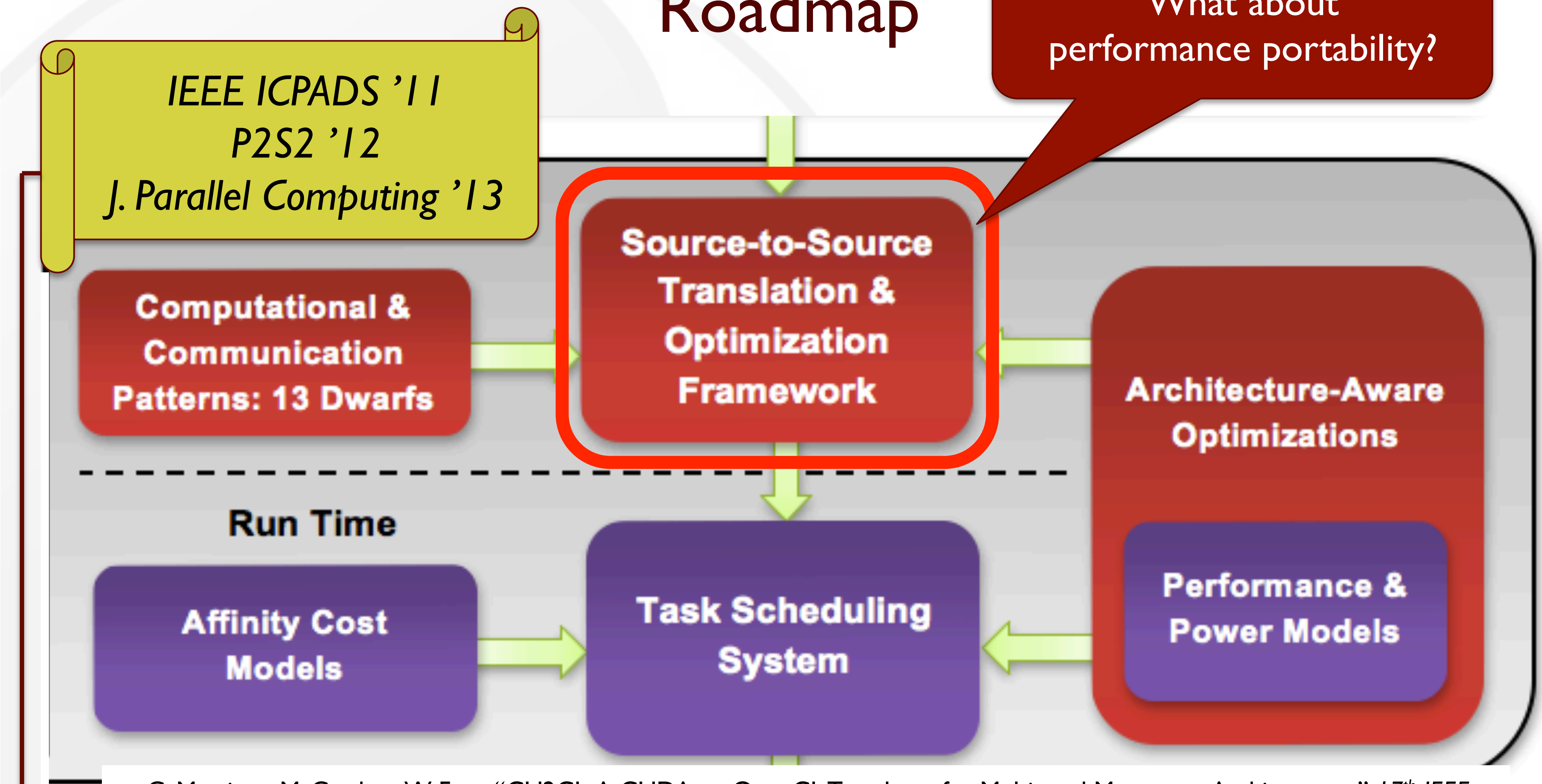


- Source: CU2CL v0.6.2b (May 2015)

Productivity = Performance + Programmability + Portability

Roadmap

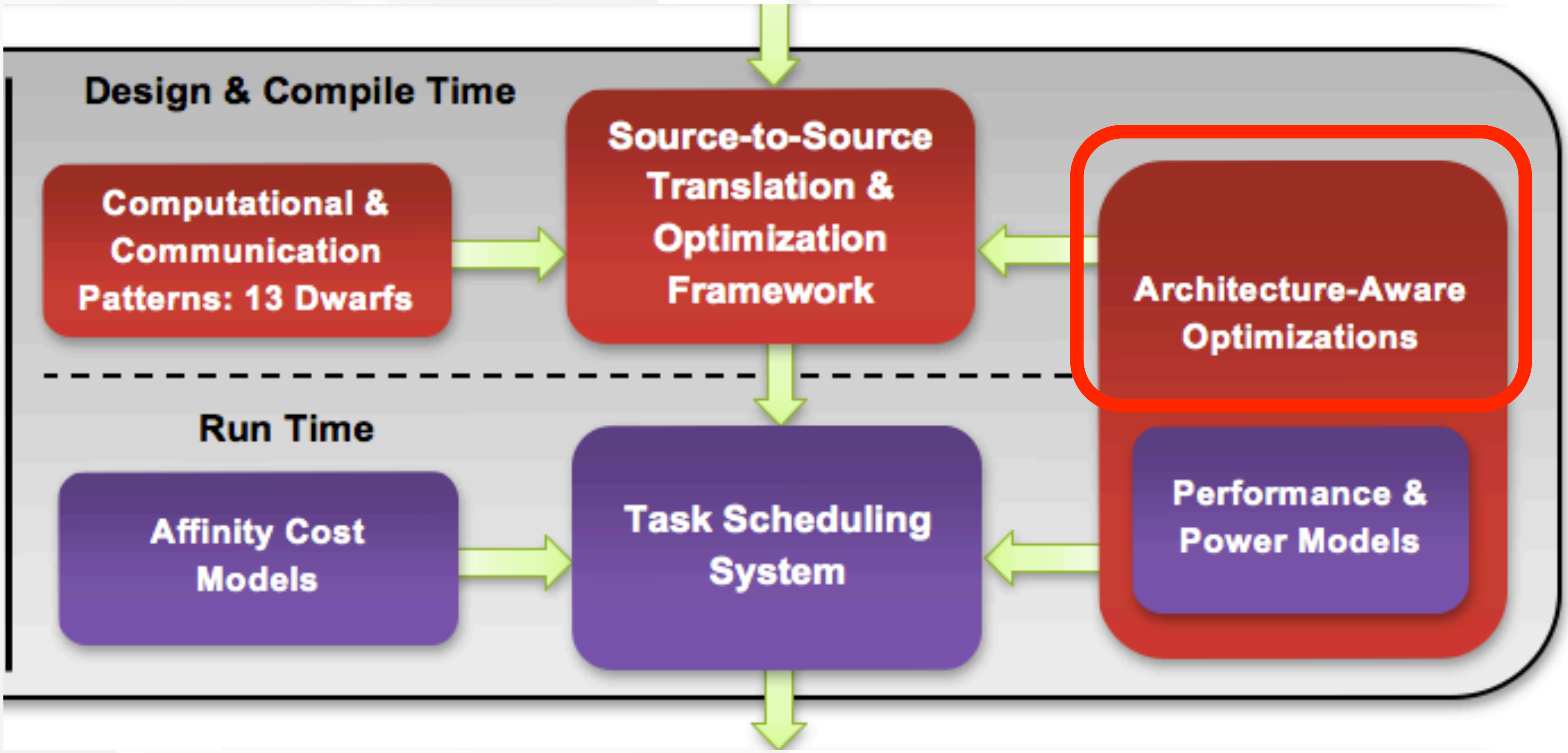
What about performance portability?



- G. Martinez, M. Gardner, W. Feng, "CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-core Architectures," *17th IEEE Int'l Conf. on Parallel & Distributed Systems (ICPADS)*, Dec. 2011.
- P. Sathre, M. Gardner, W. Feng, "Lost in Translation: Challenges in Automating CUDA-to-OpenCL Translation," *5th Int'l Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2)*, September 2012.
- M. Gardner, P. Sathre, W. Feng, and G. Martinez, "Characterizing the Challenges and Evaluating the Efficacy of a CUDA-to-OpenCL Translator," *Journal of Parallel Computing*, 39(12): 769-786, December 2013.

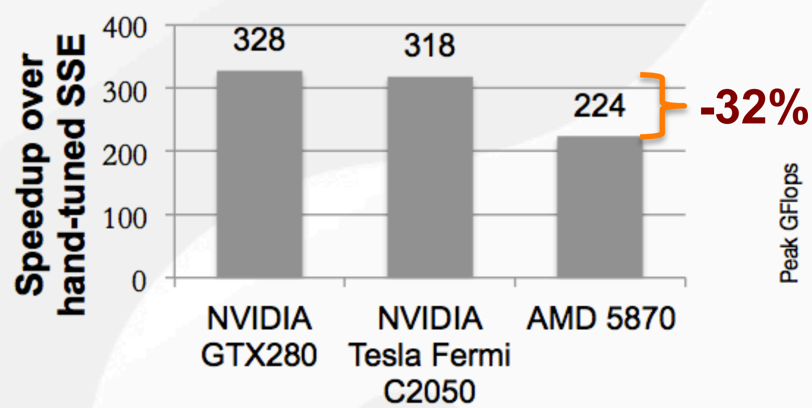
Productivity = Performance + Programmability + Portability

Roadmap

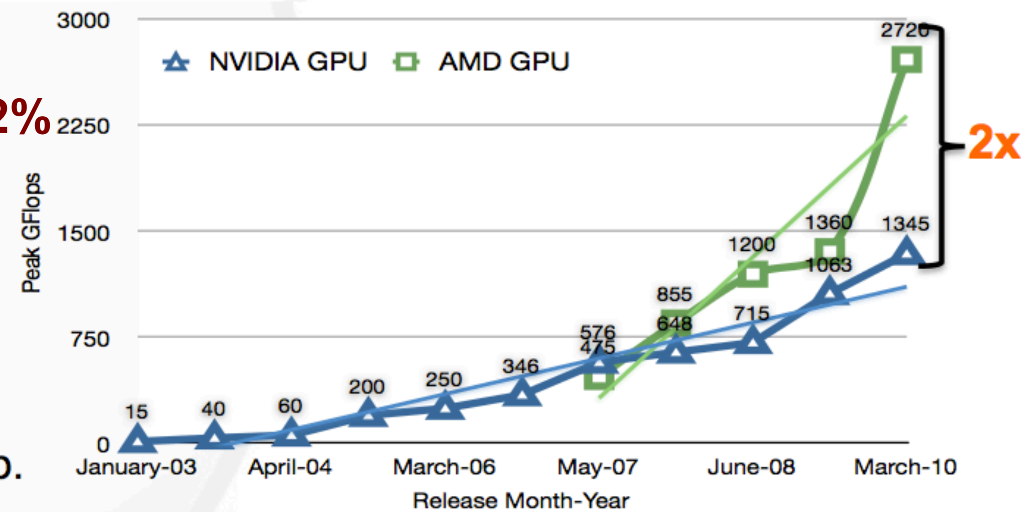


Computational Units *Not* Created Equal

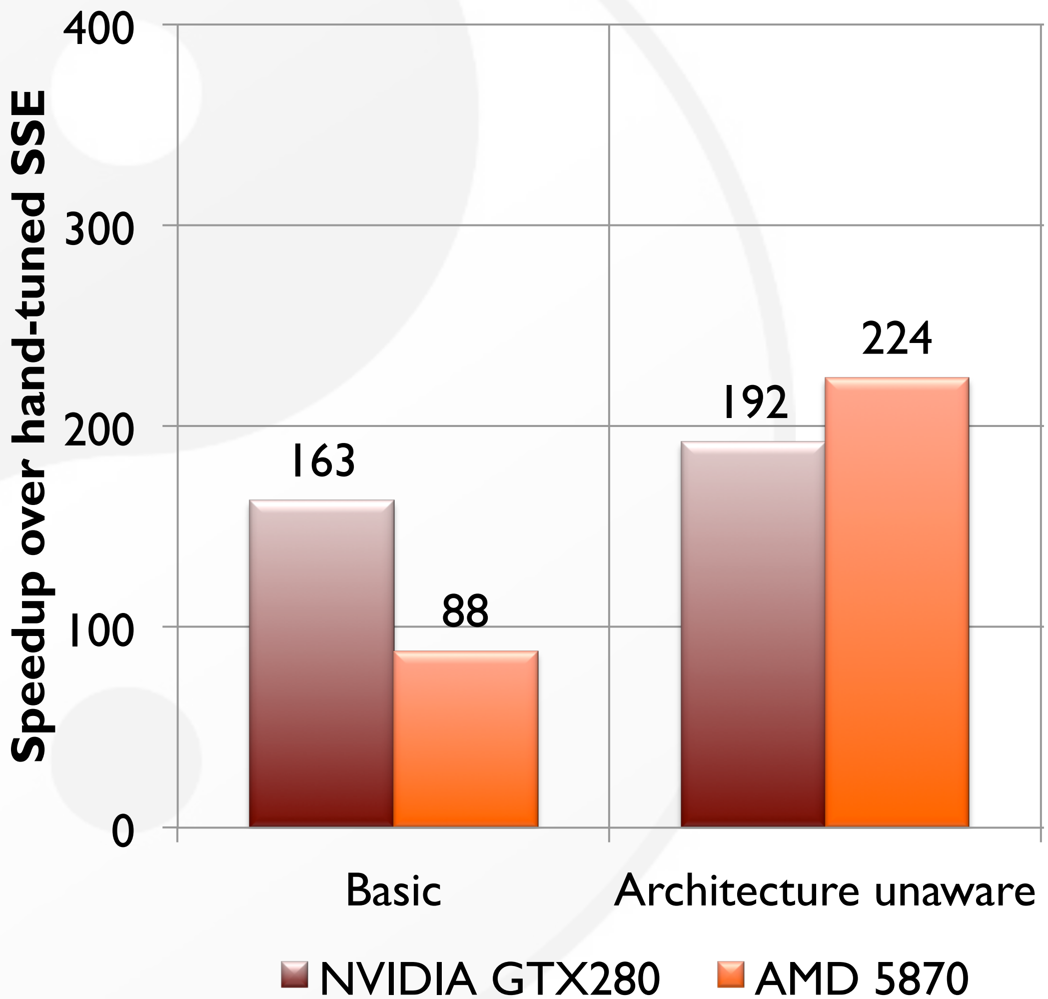
- "AMD CPU ≠ Intel CPU" and "AMD GPU ≠ NVIDIA GPU"
- Initial performance of a *CUDA-optimized* N-body dwarf



Performance of a Molecular Modeling App.

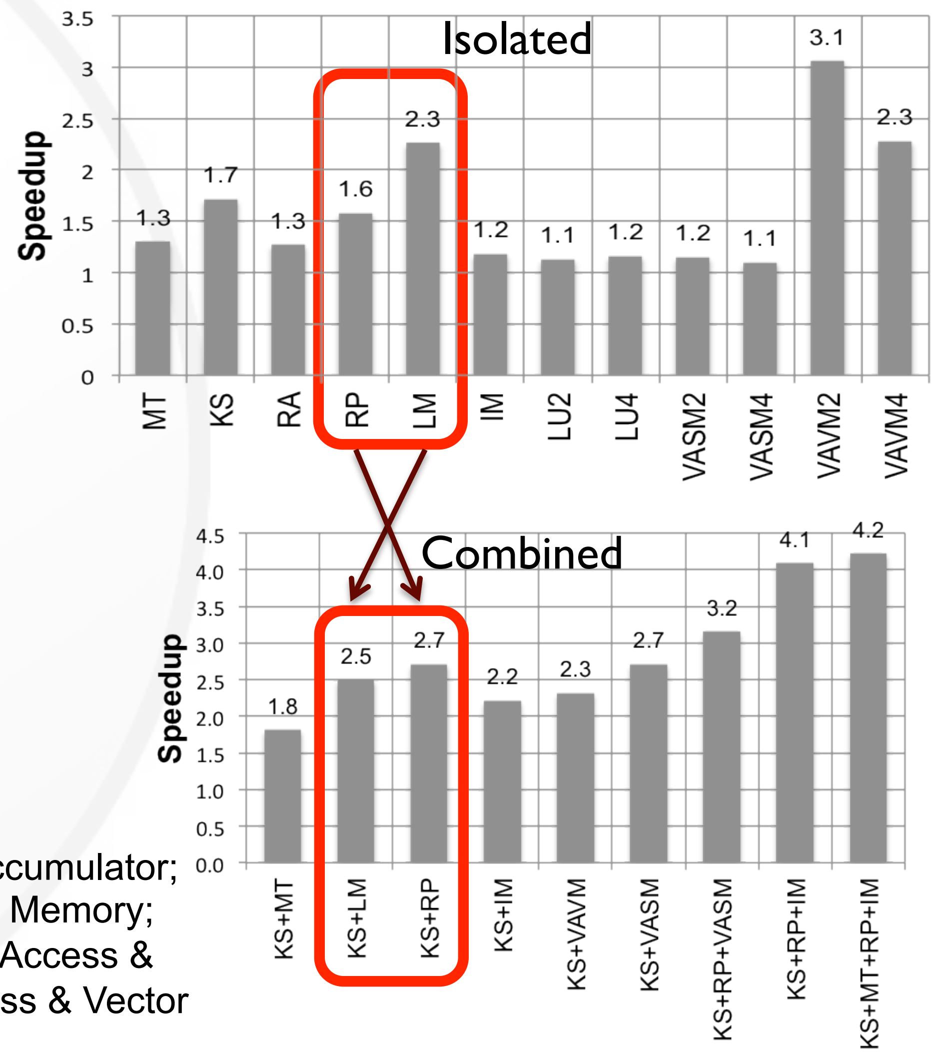


Basic Execution & Architecture-Unaware Execution



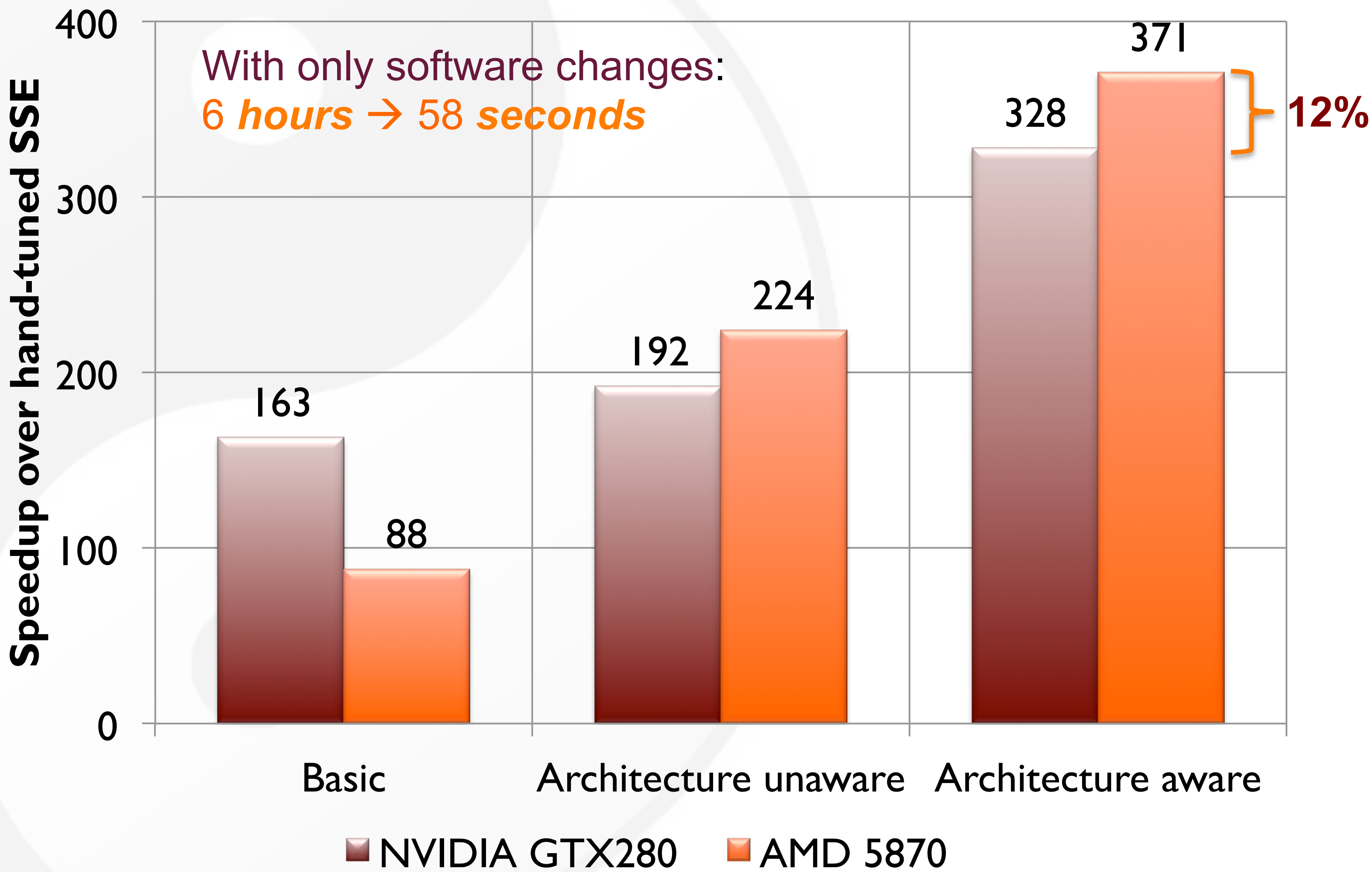
Optimization for N-body Molecular Modeling

- Optimization techniques on AMD GPUs
  - Removing conditions → kernel splitting
  - Local staging
  - Using vector types
  - Using image memory
- Speedup over basic OpenCL GPU implementation
  - Isolated optimizations
  - Combined optimizations



MT: Max Threads; **KS**: Kernel Splitting; **RA**: Register Accumulator; **RP**: Register Preloading; **LM**: Local Memory; **IM**: Image Memory; **LU{2,4}**: Loop Unrolling{2x,4x}; **VASM{2,4}**: Vectorized Access & Scalar Math{float2, float4}; **VAVM{2,4}**: Vectorized Access & Vector Math{float2, float4}

Summary: Architecture-Aware Optimization



Architecture-Aware Optimizations

… for an N-Body Code (GEM) … on AMD Radeon 5870 GPU and on NVIDIA GTX 280 GPU

Optimization Strategy	AMD	NVIDIA
Algorithm Design:		
Recomputing instead of Transferring	+ + + + +	+ + + + +
Using Host Asynchronously	+	+
Execution Configuration:		
Running Numerous Threads	+ + + + +	+ + + + +
Ensuring #Threads to be a multiple of Wavefront/Warp Size	+ + + + +	+ + + + +
Ensuring #Workgroups/Blocks to be a multiple of #Cores	+ + + + +	+ + + + +
Reducing Per-Thread Register Usage	+ + +	+ + + + +
Control Flow:		
Removing Branches	+ + +	+
Removing Divergent Branches	+ + + + +	+ + + + +
Memory Types:		
Using Registers	+ + + + +	+ + +
Using Local/Shared Memory	+ + + + +	+ + + + +
Using Constant Memory	+ + +	+ + +
Using Image/Texture Memory	+ + + + +	+
Memory Access Pattern:		
Coalescing Global Memory Accesses	+ + + + +	+ + + + +
Avoiding Partition Camping	+ + +	+ + + + +
Avoiding Bank Conflicts	+ + + + +	+ + + + +
Instruction Count:		
Using Vector Types and Operations	+ + + + +	+
Prefetching of Data from Global Memory	+ + + + +	+ + + + +
Loop Unrolling	+ + + + +	+ + + + +

For additional details, see
M. Daga*, T. Scogland*, and W. Feng, "Architecture-Aware Mapping and Optimization on a 1600-Core GPU," *17th IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS)*, Dec. 2011.

Critique: Architecture-Aware Optimizations

- Architecture
  - The Good: Two similar-generation GPUs are compared.
  - The Bad: The architectures are relatively old (circa 2008-2009).

  The architectures served the purpose of demonstrating the importance of architecture-aware optimizations.
- Workload
  - The Good: A real application code (i.e., GEM – an n-body molecular modeling code) is optimized
  - The Bad: Only one application code is optimized.

  What would be the impact of the architecture-aware optimizations be on other application workloads?
- Manual Optimization → Automatic Optimization

Critique: Architecture-Aware Optimizations

Address first two now …

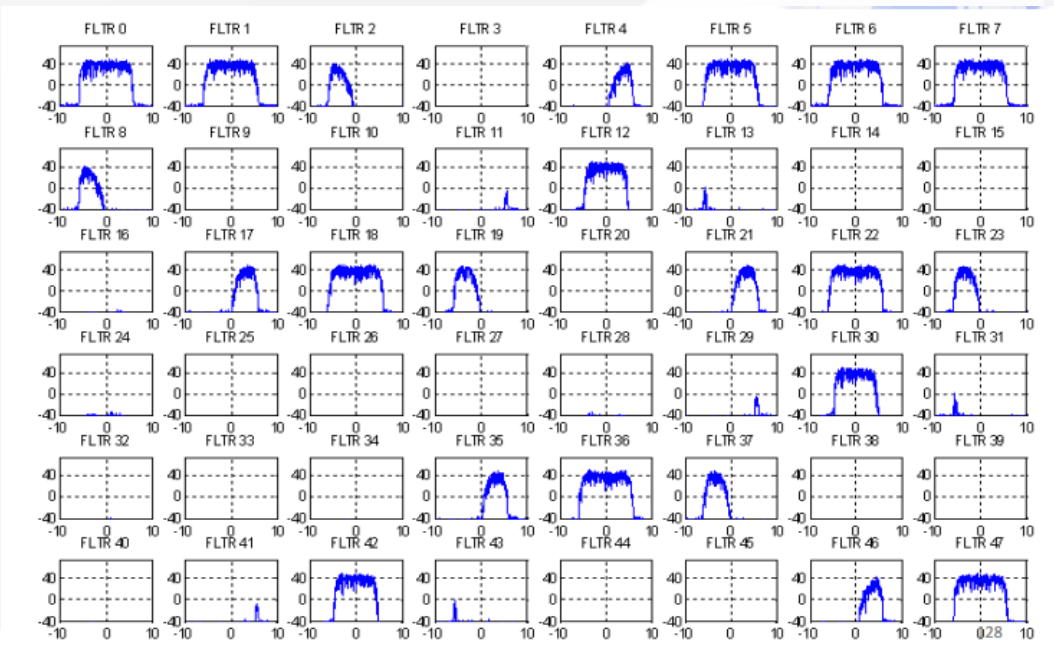
- Architecture
  - The Good: Two similar-generation GPUs are compared.
  - The Bad: The architectures are relatively old (circa 2008-2009).

  The architectures served the purpose of demonstrating the importance of architecture-aware optimizations.
- Workload
  - The Good: A real application code (i.e., GEM – an n-body molecular modeling code) is optimized
  - The Bad: Only one application code is optimized.

  What would be the impact of the architecture-aware optimizations be on other application workloads?
- Manual Optimization → Automatic Optimization

FFT: Fast Fourier Transform

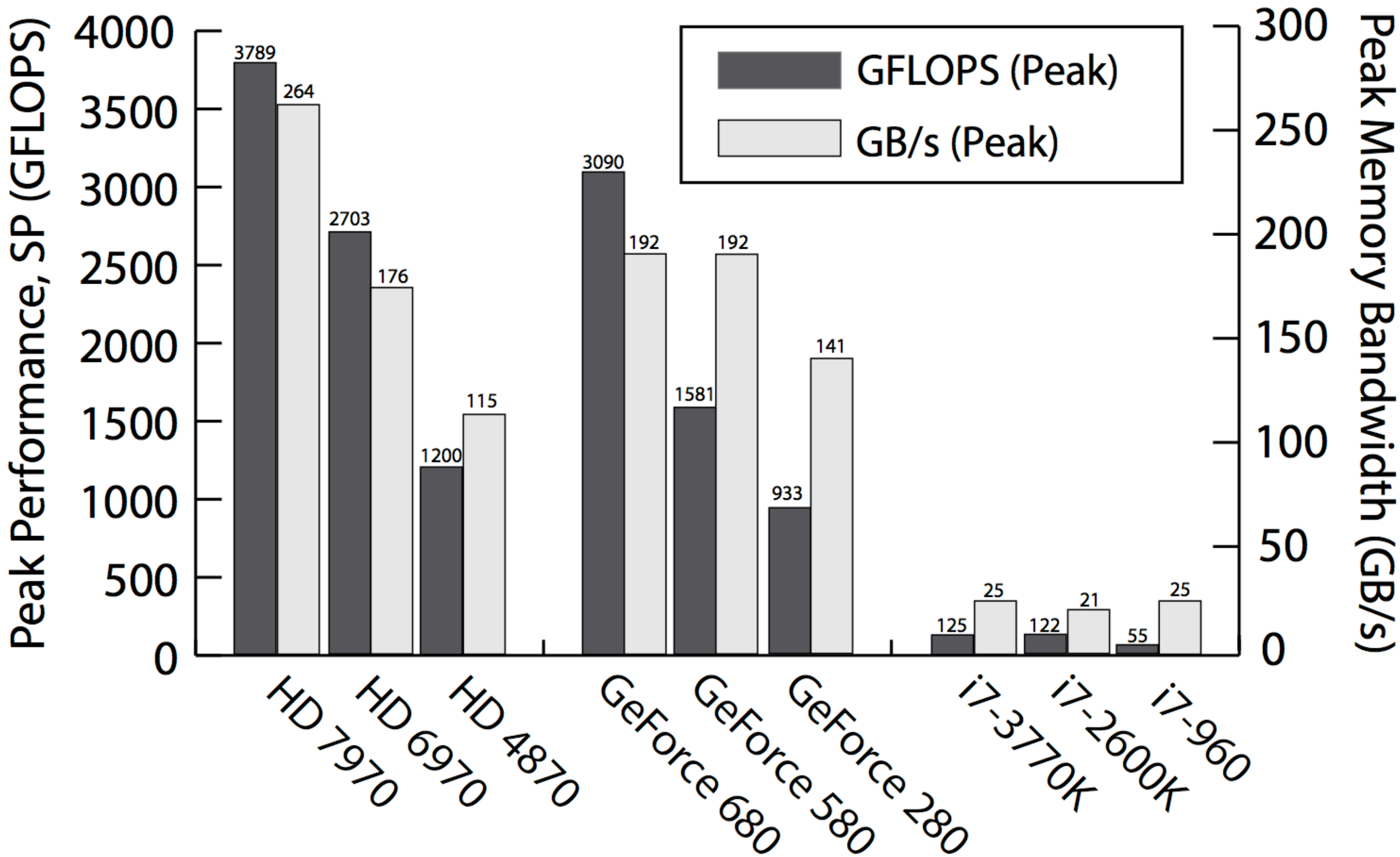
- A spectral method that is a critical building block across many disciplines



Peak Specifications of GPU Cards

Device	Cores	Peak Performance (GFLOPS)	Global Memory Bandwidth (GB/s)	Register Size per CU (kB)	LDS Size per CU (kB)	Core Clock (MHz)	Memory Clock (MHz)	Max TDP (Watts)
Radeon HD 6970 GPU	1536	2703	176	256	32	880	1375	250
Radeon HD 7970 GPU	2048	3788	264	256	64	925	1375	250
Tesla C2075 GPU	448	1288	144	32	48	1147	1666	225
Tesla K20c GPU	2496	4106	208	64	48	705	2600	225

Peak SP Performance & Peak Memory Bandwidth



Summary of Optimizations

Codename	Name	Description
LM-CT	Local Memory (Computation, No Transpose)	Data elements are loaded into local memory for computation. The communication step is avoided by algorithm reorganization [8].
LM-CC	Local Memory (Computation and Communication)	All data elements are preloaded into local memory. Computation is performed in local memory, while registers are used for scratchpad communication.
LM-CM	Local Memory (Communication Only)	Data elements are loaded into local memory only for communication. Threads swap data elements solely in local memory. This optimization requires only $N\times$ sz(float) local memory by transposing each dimension of a floatn vector one dimension at a time.
CM-{K,L}	Constant Memory {Kernel, Literal}	The twiddle multiplication stage can be pre-computed on the host and stored in constant memory for fast look up. This saves two transcendental single-precision operations at the cost of a cached memory access. CM-K refers to the usage of constant memory as a kernel argument, while CM-L refers to a static global declaration in the OpenCL kernel.
RP	Register Preloading	All data elements are first preloaded onto the register file of the respective GPU. Computation is facilitated solely on registers.
CGAP	Coalesced Global Access Pattern	Threads in a wavefront access memory contiguously, e.g. the kth thread accesses memory element, k.
CSE	Common Subexpression Elimination	A traditional optimization that collapses identical expressions in order to save computation. This optimization may increase register live time, therefore, increasing register pressure.
IL	(Function) Inlining	A function's code body is inserted in place of a function call. It is used primarily for functions that are frequently called.
LU	Loop Unrolling	A loop is explicitly rewritten as an identical sequence of statements without the overhead of loop variable comparisons.
VASM{2,4,8,16}	Vector Access, Scalar Math float{2,4,8,16}	Data elements are loaded as the listed vector type. Arithmetic operations are scalar (float $\times$ float).
VAVM{2,4,8,16}	Vector Access, Vector Math float{2,4,8,16}	Data elements are loaded as the listed vector type. The arithmetic operations are vectorized with the sizes listed, (floatn $\times$ floatn).
SHFL	Shuffle	The transpose stage in FFT is performed entirely in registers eliminating the use of local memory. This optimization is only specific to NVIDIA Tesla K20c

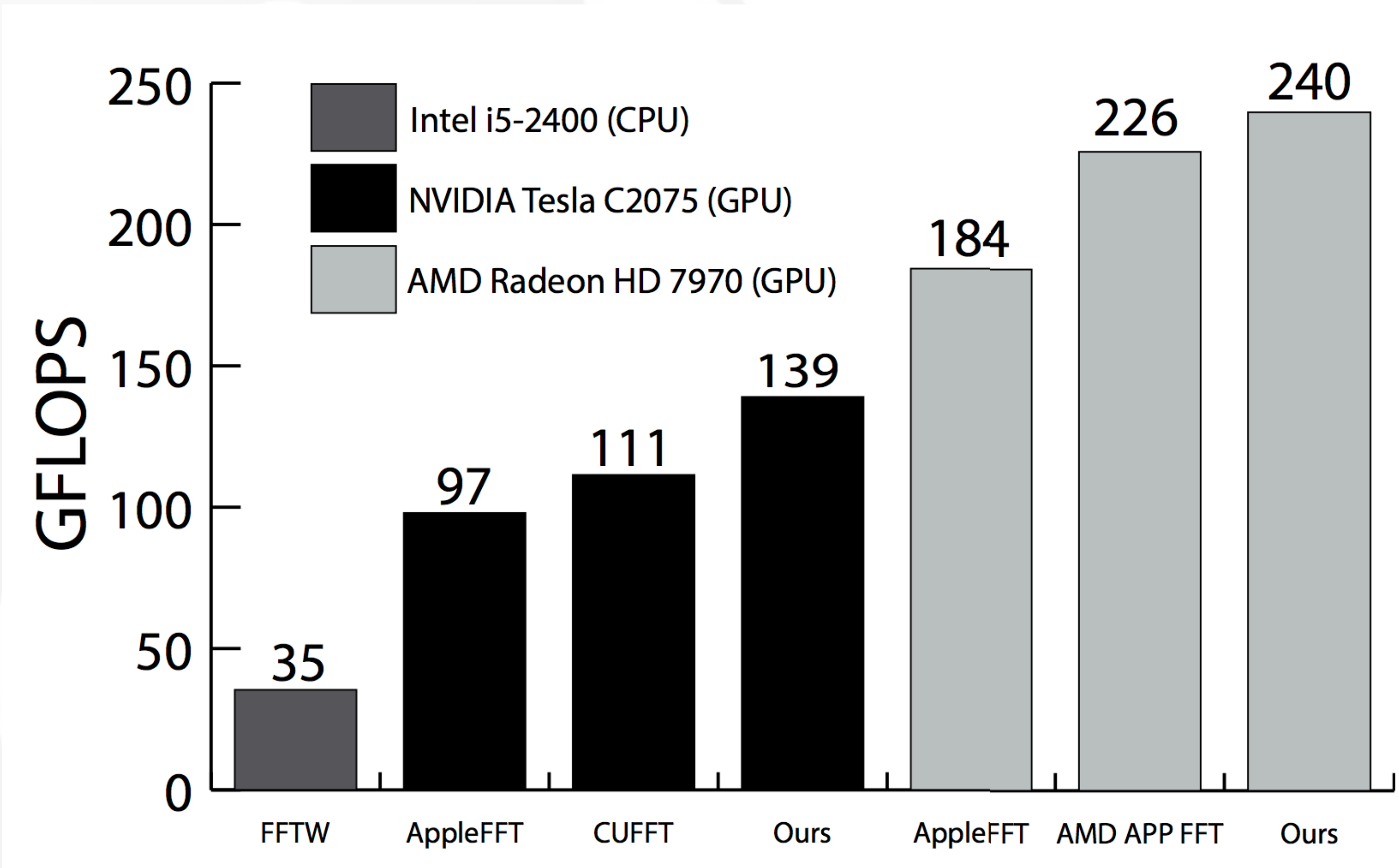
Optimizations

- RP: Register Preloading
  - All data elements are first preloaded into the register file before use.
  - Computation facilitated solely on registers
- LM-CM: Local Memory (Communication Only)
  - Data elements are loaded into local memory only for communication
  - Threads swap data elements solely in local memory
- CGAP: Coalesced Global Access Pattern
  - Threads access memory contiguously
- VASM{2|4}: Vector Access, Scalar Math, float{2|4}
  - Data elements are loaded as the listed vector type.
  - Arithmetic operations are scalar (float × float).
- CM-K: Constant Memory for Kernel Arguments
  - The twiddle multiplication state of FFT is precomputed on the CPU and stored in GPU constant memory for fast look-up.

Architecture-Optimized FFT

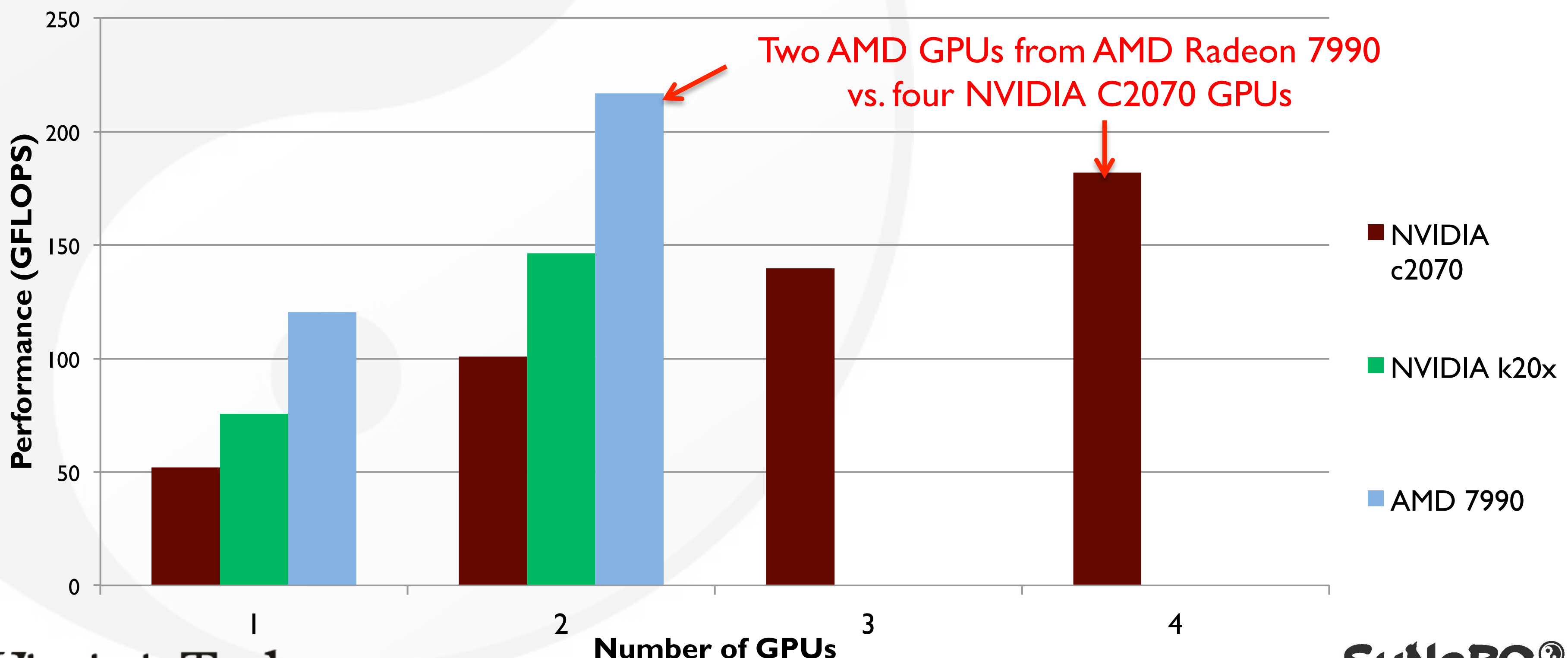
Device	Sample Size	Baseline (GFLOPS)	Optimal (GFLOPS)	Speedup	Speedup over FFTW	Optimal Set
Intel i5-2400 (FFTW)	16		36			
	64		43			
	256		48			
Radeon HD 6970	16	12	174	14.5x	4.8x	RP + LM-CM + CGAP + **VASM4** + CM-K
	64	14	257	18.4x	6.0x	RP + LM-CM + CGAP + **VASM4** + CM-K
	256	11	346	**31.5x**	7.2x	RP + LM-CM + CGAP + **VASM2** + CM-K
Radeon HD 7970	16	36	240	6.7x	6.7x	RP + LM-CM + CGAP + **VASM4** + CM-K
	64	23	366	15.9x	8.5x	RP + LM-CM + CGAP + **VASM2** + CM-K
	256	24	437	18.2x	**9.1x**	RP + LM-CM + CGAP + **VASM2** + CM-K
Tesla C2075	16	37	139	3.7x	3.9x	RP + LM-CM + CGAP + **VASM2** + CM-K
	64	69	200	2.9x	4.7x	RP + LM-CM + CGAP + **VASM4** + CM-K
	256	60	177	3.0x	3.7x	RP + LM-CM + CGAP + **VASM2** + CM-K
Tesla K20c	16	54	183	3.4x	5.1x	RP + LM-CM + CGAP + **VASM2** + CM-K
	64	99	265	2.7x	6.2x	RP + LM-CM + CGAP + **VASM4** + CM-K
	256	95	280	2.9x	5.8x	RP + LM-CM + CGAP + **VASM2** + CM-K

Architecture-Optimized FFT
(Batched, Single Precision, 1-D, 16-pt)



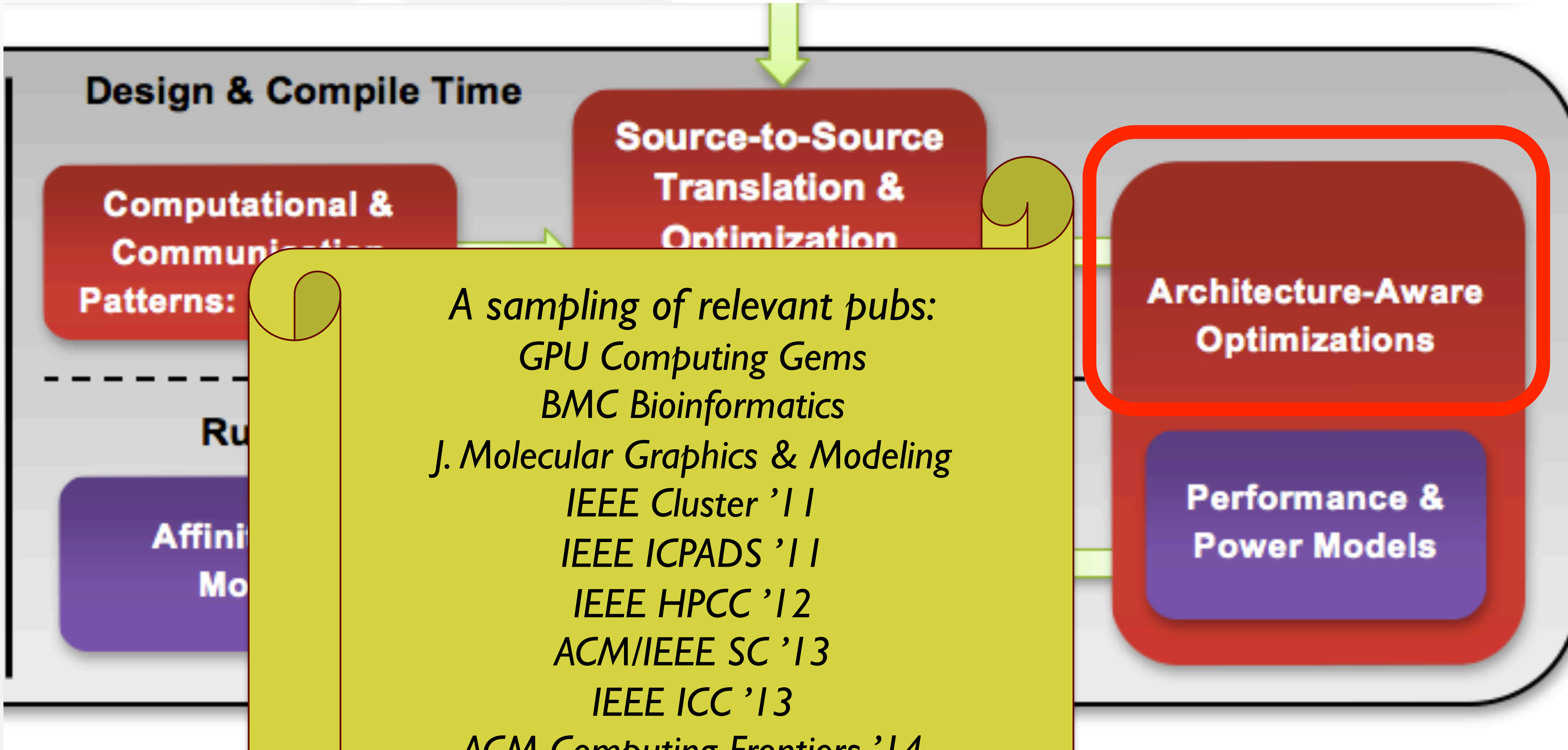
Multi-GPU Scaling

- Near-linear performance scaling using multiple GPUs.
  - Domain decomposition, with each domain partition residing on one GPU for duration of simulation (only ghost cells had to be exchanged on each iteration).
  - One control CPU thread per GPU.
  - PGI 14.1 compiler can generate code for AMD GPUs in addition to NVIDIA.



Productivity = Performance + Programmability + Portability

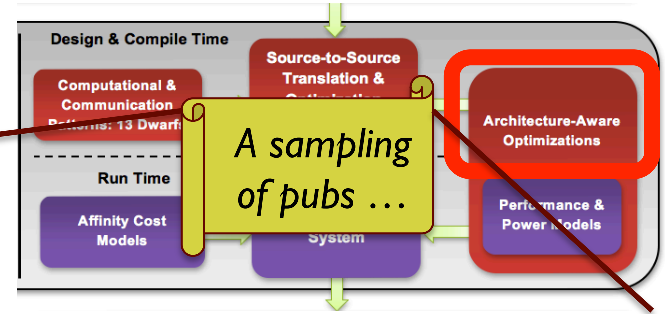
Roadmap



A sampling of relevant pubs:
GPU Computing Gems
BMC Bioinformatics
J. Molecular Graphics & Modeling
IEEE Cluster '11
IEEE ICPADS '11
IEEE HPCC '12
ACM/IEEE SC '13
IEEE ICC '13
ACM Computing Frontiers '14

Productivity = Performance + Programmability + Portability

Roadmap



- R. Anandakrishnan, T. Scogland, A. Fenley, J. Gordon, W. Feng, A. Onufriev, "Accelerating Electrostatic Surface Potential Calculation with Multiscale Approximation on Graphics Processing Units," *J. Molecular Graphics and Modelling*, 28(8): 904-910, June 2010.
- W. Feng, Y. Cao, D. Patnaik, N. Ramakrishnan, "Temporal Data Mining for Neuroscience" in *GPU Computing Gems*, Elsevier/Morgan-Kaufmann, Feb. 2011.
- M. Daga, T. Scogland, W. Feng, "Towards Accelerating Molecular Modeling via Multi-Scale Approximation on a GPU," *IEEE Int'l Conf. on Computational Advances in Bio and Medical Sciences (ICCABS)*, Feb. 2011.
- M. Daga, A. Aji, W. Feng, "On the Efficacy of a Fused CPU+GPU Processor for Parallel Computing," *Symp. on Application Accelerators in High-Performance Computing (SAAHPC)*, Jul. 2011.
- M. Elteir, H. Lin, and W. Feng, "Performance Characterization and Optimization of Atomic Operations on AMD GPUs," *IEEE Cluster*, Sept, 2011.
- M. Daga, T. Scogland, W. Feng, "Architecture-Aware Mapping and Optimization on a 1600-Core GPU," *IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS)*, Dec. 2011.
- M. Daga and W. Feng, "Multi-Dimensional Characterization of Electrostatic Surface Potential Computation on Graphics Processors," *BMC Bioinformatics*, 13(Suppl 5):S4, Apr. 2012.
- F. Ji, A. Aji, J. Dinan, D. Buntinas, P. Balaji, R. Thakur, W. Feng, X. Ma, "DMA-Assisted, Intranode Communication in GPU-Accelerated Systems," *IEEE International Conference on High-Performance Computing and Communications*, Jun. 2012.
- C. del Mundo, V. Adhinarayanan, W. Feng, "Accelerating FFT for Wideband Channelization." *IEEE ICC '13*, Jun. 2013.
- C. del Mundo, W. Feng. "Enabling Efficient Intra-Warp Communication for Fourier Transforms in a Many-Core Architecture", *SC|13*, Nov. 2013.
- C. del Mundo, W. Feng. "Towards a Performance-Portable FFT Library for Heterogeneous Computing," *ACM Computing Frontiers*, May 2014.

Critique: Architecture-Aware Optimizations

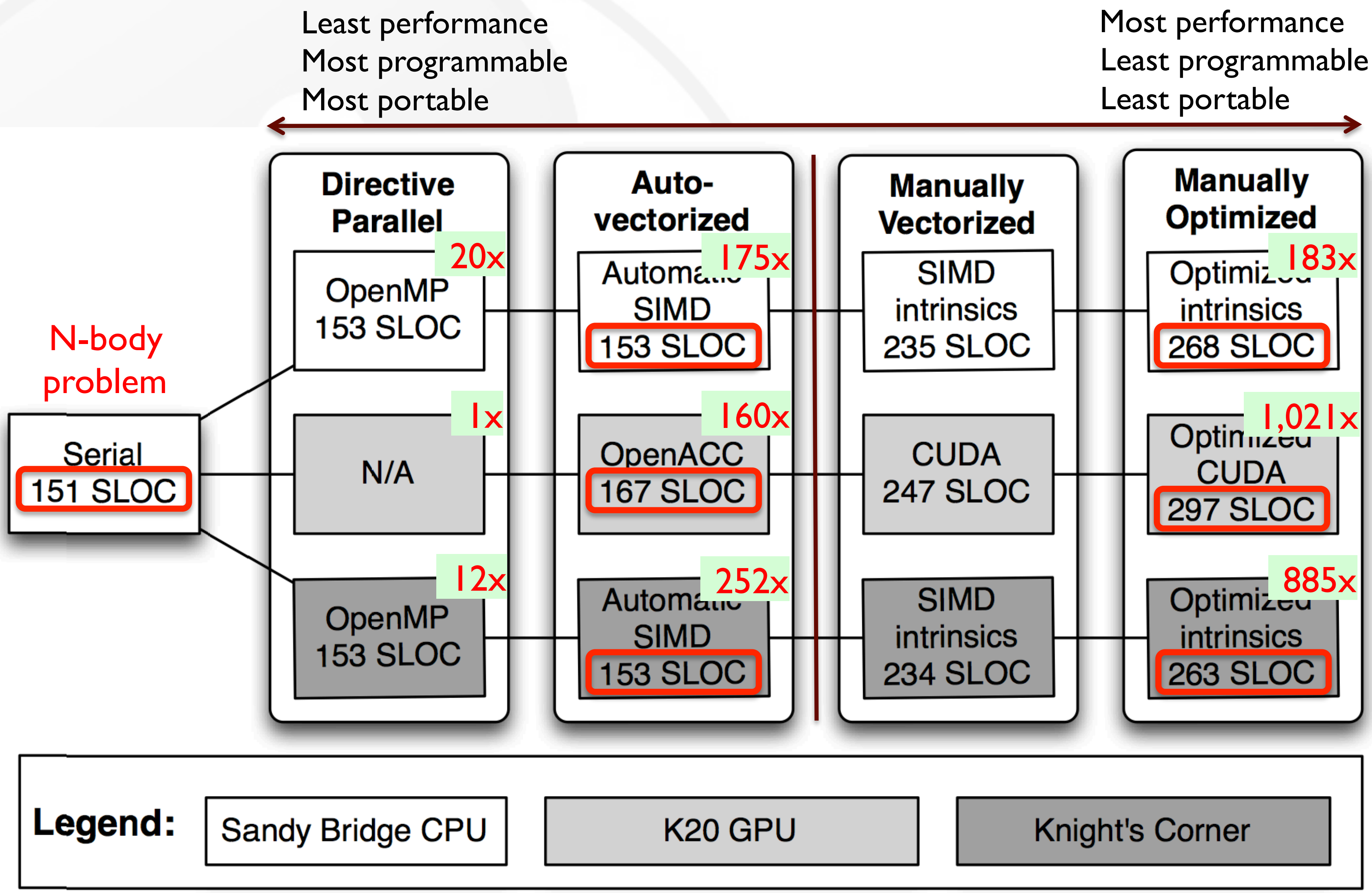
- Architecture
  - The Good: Two similar-generation GPUs are compared.
  - The Bad: The architectures are relatively old (circa 2008-2009).

  The architectures served the purpose of demonstrating the importance of architecture-aware optimizations.
- Workload
  - The Good: A real application code (i.e., GEM – an n-body molecular modeling code) is optimized
  - The Bad: Only one application code is optimized.

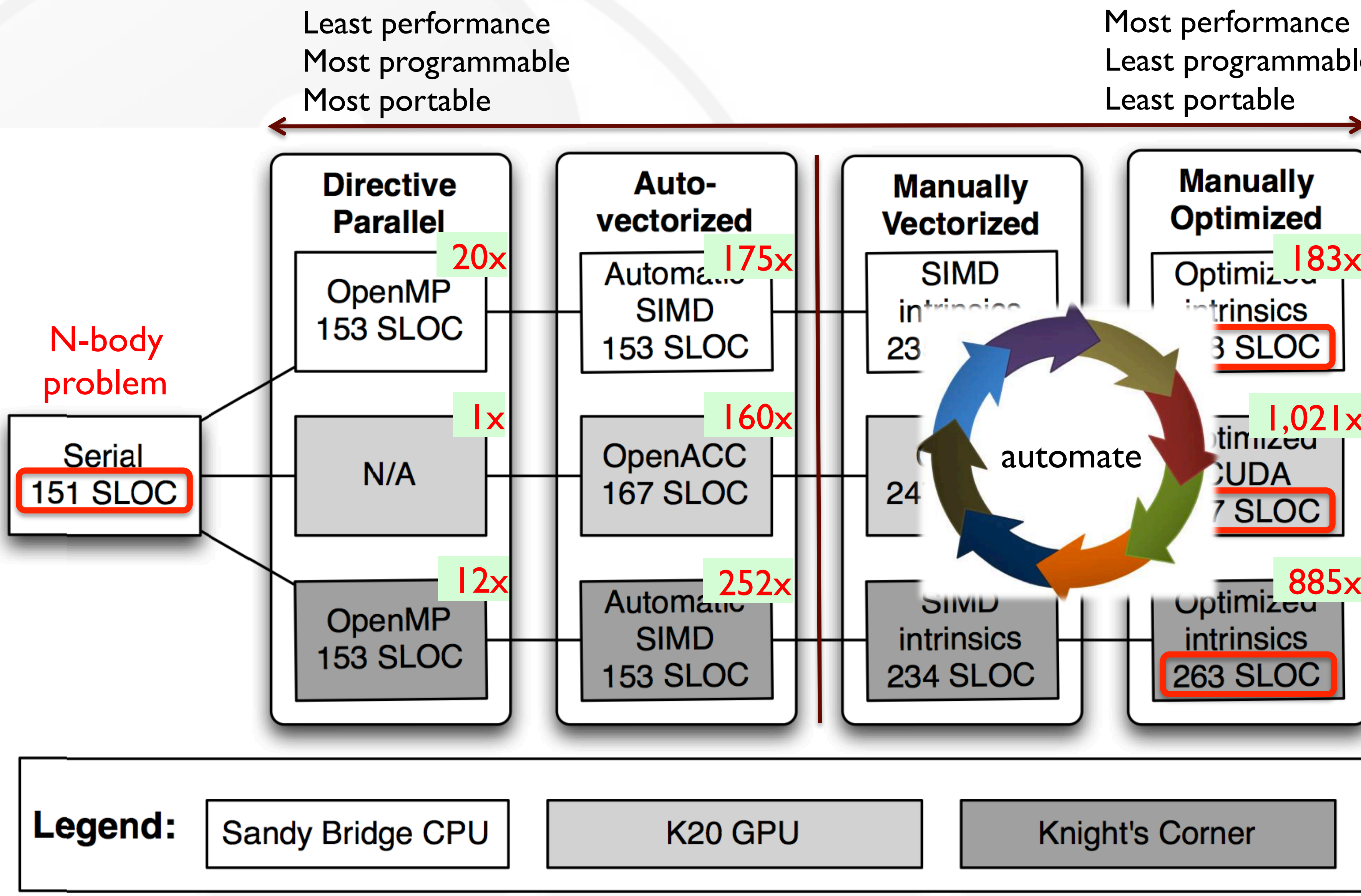
  What would be the impact of the architecture-aware optimizations be on other application workloads?
- Manual Optimization → Automatic Optimization

Still to do …

"Productivity = Performance + Programmability + Portability"



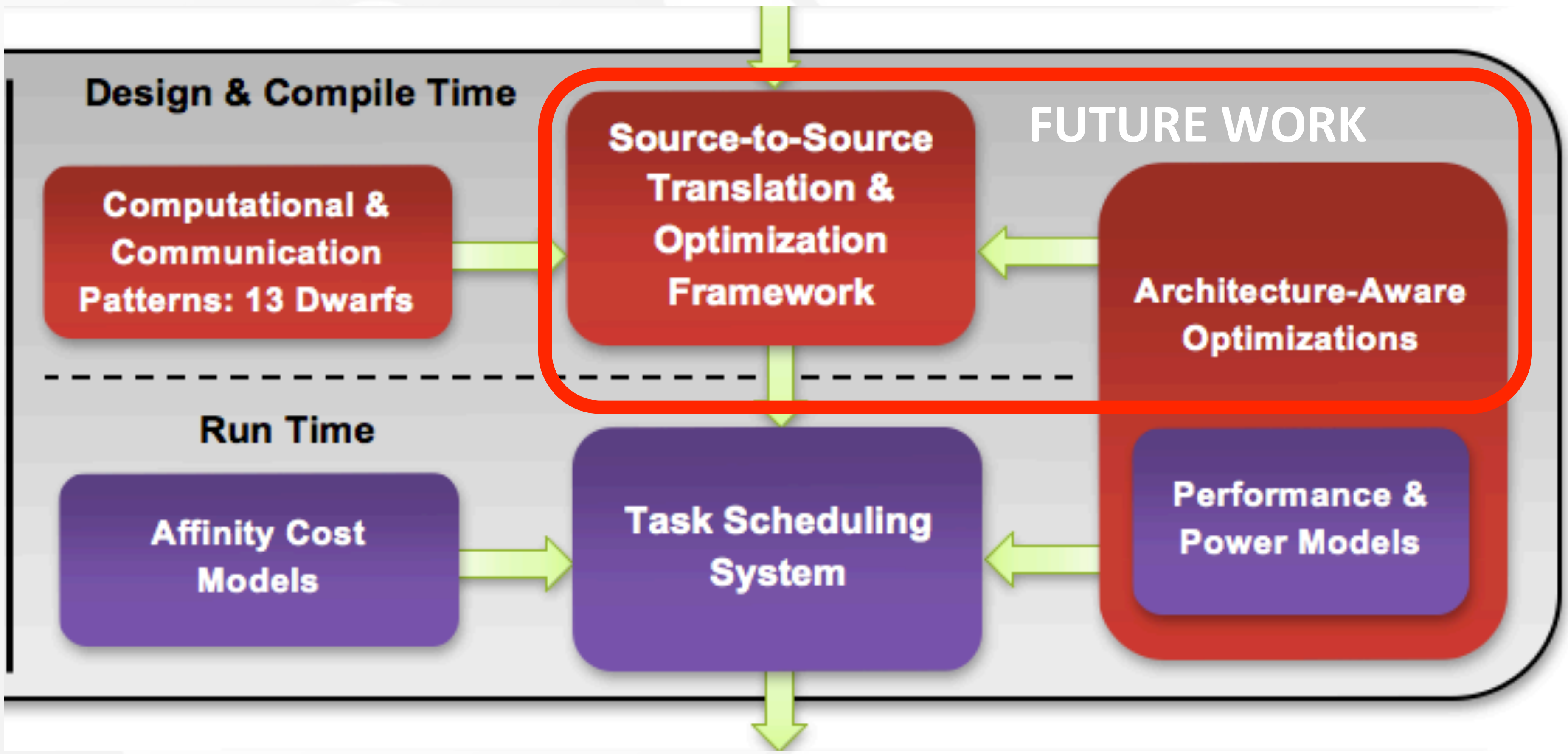
“Productivity = Performance + Programmability + Portability”



Example: Stanford has 1,000,000 CUDA SLOC that they need to translate to OpenACC/OpenMP or OpenCL

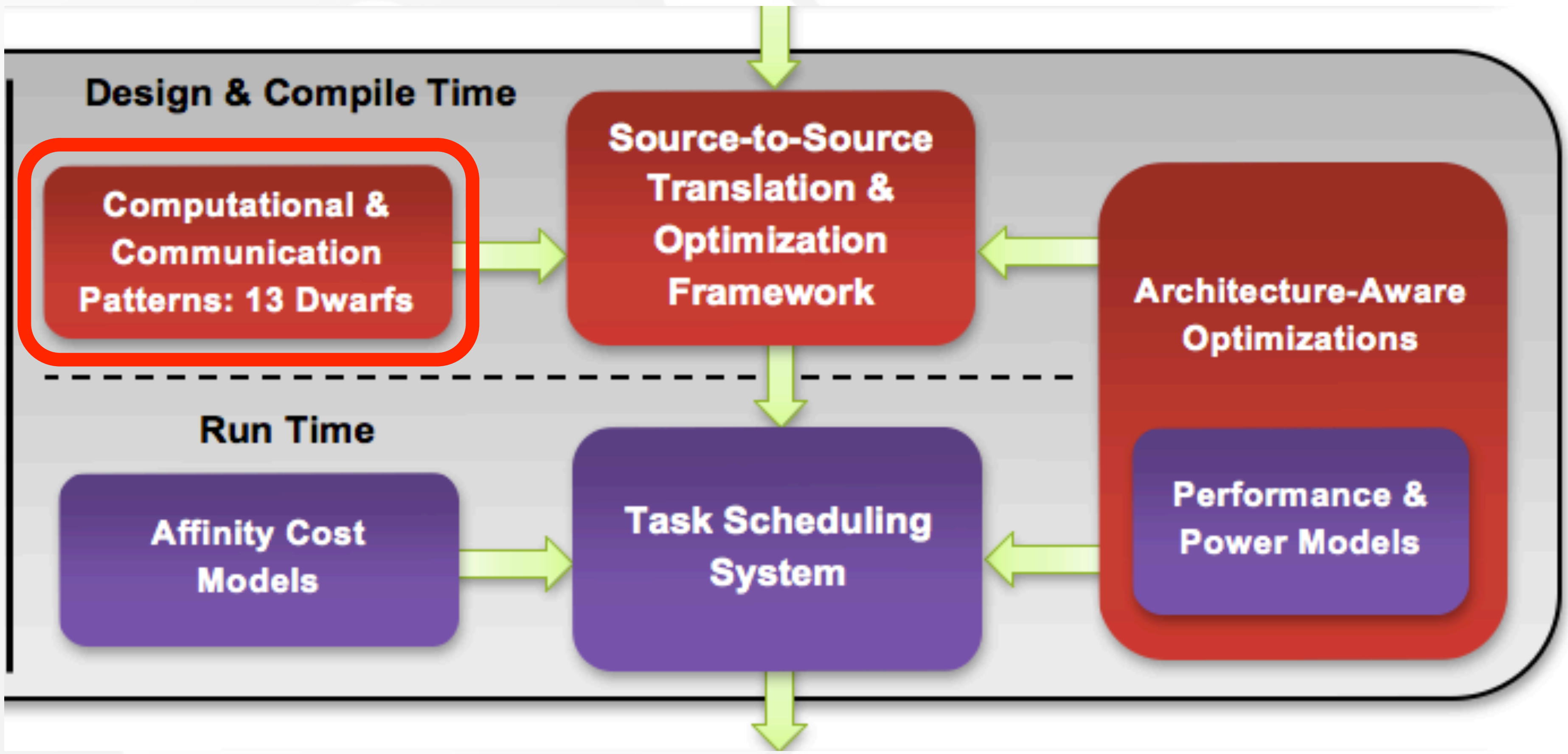
Productivity = Performance + Programmability + Portability

Roadmap



Productivity = Performance + Programmability + Portability

Roadmap



Paying For Performance

- "The free lunch is over..." †
  - Programmers can no longer expect substantial increases in single-threaded performance.
  - The burden falls on developers to exploit parallel hardware for performance gains.

- How do we lower the cost of concurrency?

† H. Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's Journal*, 30(3), March 2005. (Updated August 2009.)

The Berkeley View †

- Traditional Approach
  - Applications that target existing hardware and programming models
- Berkeley Approach
  - Hardware design that keeps future applications in mind
  - Basis for future applications? 13 computational dwarfs

A computational dwarf is a pattern of communication & computation that is common across a set of applications.

Sparse Linear Algebra
Dense Linear Algebra
Spectral Methods
Structured Grids

N-Body Methods
Unstructured Grids
Monte Carlo → MapReduce

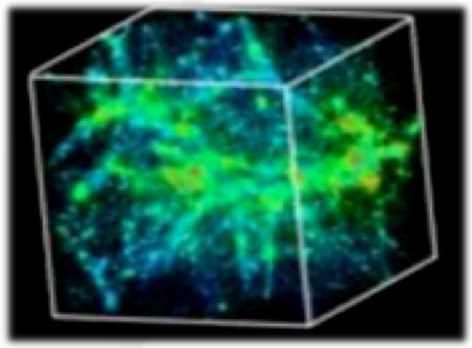
and

Combinational Logic
Graph Traversal
Dynamic Programming
Backtrack & Branch+Bound
Graphical Models
Finite State Machine

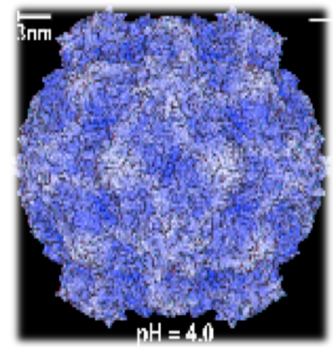
† Asanovic, K., et al. *The Landscape of Parallel Computing Research: A View from Berkeley*. Tech. Rep. UCB/EECS-2006-183, University of California, Berkeley, Dec. 2006.

Example of a Computational Dwarf: N-Body

- Computational Dwarf: *Pattern of computation & communication … that is common across a set of applications*
- N-body problems are studied in
  - Cosmology, particle physics, biology, and engineering
- All have similar structures
- An N-body benchmark can provide meaningful insight to people in all these fields
- Optimizations may be generally applicable as well



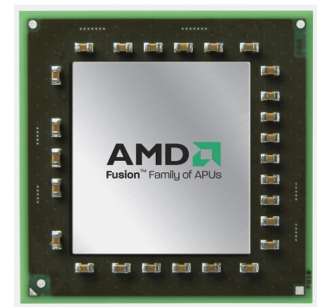
RoadRunner Universe: Astrophysics



GEM: Molecular Modeling

First Instantiation: OpenDwarfs
(formerly "OpenCL and the 13 Dwarfs")

- Goal
  - Provide common algorithmic methods, i.e., dwarfs, in a language that is "write once, run anywhere" (CPU, GPU, or even FPGA), i.e., OpenCL

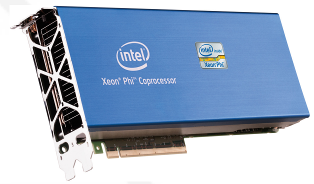
- Part of a larger umbrella project (2008-2018), funded by the NSF Center for High-Performance Reconfigurable Computing (CHREC)

CHREC
NSF Center for High-Performance Reconfigurable Computing

AMD HARRIS XILINX ALTERA Los Alamos NATIONAL LABORATORY

Experimental Setup: Hardware

Multicore CPU:
AMD Opteron 6272



Intel Many Integrated Core (MIC) Co-processor:
Intel Xeon Phi P1750

Accelerated Processing Units (APUs):

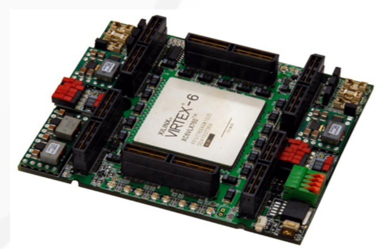
a) AMD Llano A8-3850 (Llano)

- ***Multicore CPU***: *AMD Llano A8-3850*
- ***Integrated GPU (iGPU)***: AMD Radeon HD 6550D

b) AMD Llano A10-5800K (Trinity)

- ***Multicore CPU***: *AMD Llano A10-5800K*
- ***Integrated GPU (iGPU)***: AMD Radeon HD 7660D

GPU (discrete):
AMD Radeon HD 7970



Field Programmable Gate Array (FPGA):
Xilinx Virtex-6 LX760, 1 GB DRAM, PCIx connectivity

K. Krommydas, W. Feng, … "On the Characterization of OpenCL Dwarfs on Fixed and Reconfigurable Platforms," *25th IEEE Int'l Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, June 2014. Best Paper Finalist

Experimental Setup: Software

Benchmarks

OpenDwarfs

- GEM (N-body methods)
- NW (Dynamic programming)
- SRAD (Structured grids)
- BFS (Graph traversal)

Host systems

AMD CPU/GPU/APU: 64-bit Debian Linux 7.0 (Kernel 2.6.37), AMD APP SDK 2.8, AMD Catalyst 13.1

Intel Xeon Phi: CentOS 6.3, Intel OpenCL SDK XE 2013

FPGA: Ubuntu 12.04, Xilinx ISE 12.4

Profiling tools

AMD CPU/GPU/APU:

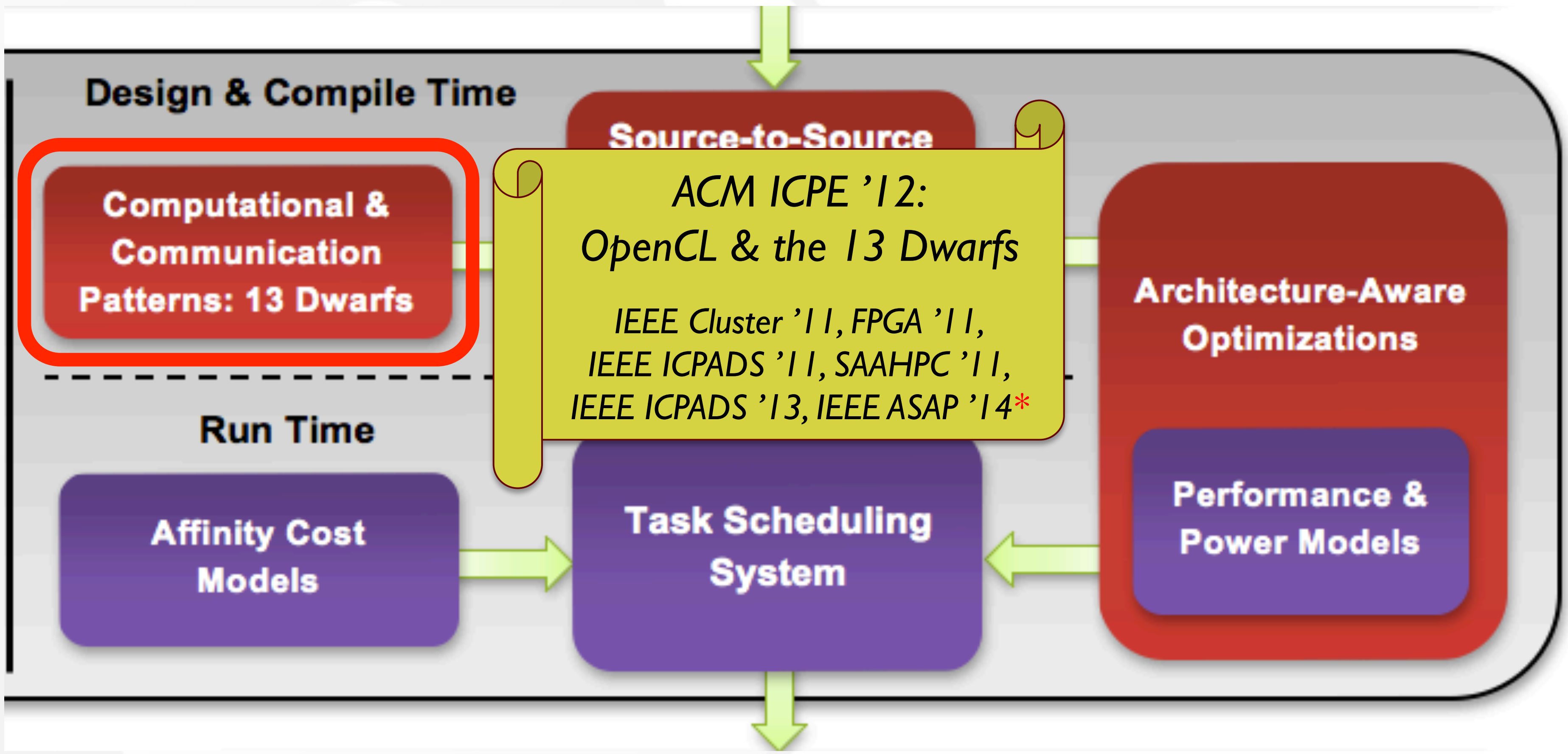
AMD CodeXL 1.3

Intel Xeon Phi:

Intel Vtune Amplifier XE 2013

Productivity = Performance + Programmability + Portability

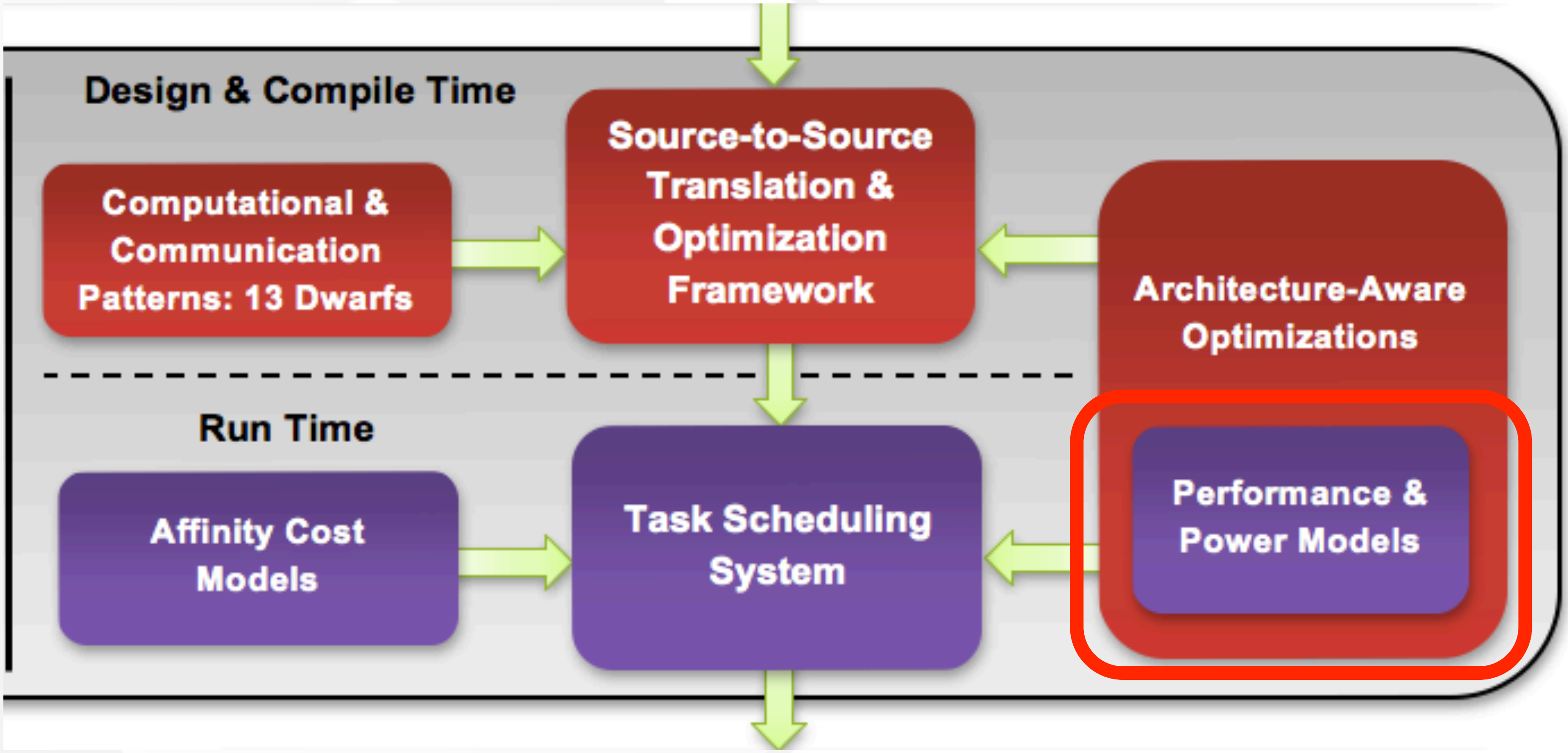
Roadmap



* K. Krommydas, W. Feng, M. Owaida, C. Antonopoulos, N. Bellas, "On the Characterization of OpenCL Dwarfs on Fixed and Reconfigurable Platforms, *25th IEEE Int'l Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, June 2014. **Best Paper Finalist**

Productivity = Performance + Programmability + Portability

Roadmap



Performance & Power Modeling

- Goals
  - Robust framework
  - Very high accuracy (Target: < 5% prediction error)
  - Identification of portable predictors for performance and power
  - Multi-dimensional characterization
    - Performance → sequential, intra-node parallel, inter-node parallel
    - Power → component level, node level, cluster level

Problem Formulation:

LP-Based Energy-Optimal DVFS Schedule

- Definitions
  - A DVFS system exports *n* { (f_i, P_i) } settings.
  - T_i : total execution time of a program running at setting *i*
- Given a program with deadline *D*, find a DVS schedule (t_1*, …, t_n*) such that
  - If the program is executed for t_i seconds at setting *i*, the total energy usage E is minimized, the deadline D is met, and the required work is completed.

$$\min E = \sum_i P_i \cdot t_i$$

subject to

$$\sum_i t_i \leq D$$
$$\sum_i t_i / T_i = 1$$
$$t_i \geq 0$$

Single-Coefficient β Performance Model

- Our Formulation
  - Define the relative performance slowdown δ as

    $T(f) / T(f_{MAX}) - 1$
  - Re-formulate two-coefficient model as a single-coefficient model:

$$\frac{T(f)}{T(f_{max})} = \beta \cdot \frac{f_{max}}{f} + (1 - \beta)$$

with

$$\beta = \frac{W_{cpu}}{W_{cpu} + T_{mem} \cdot f_{max}}$$

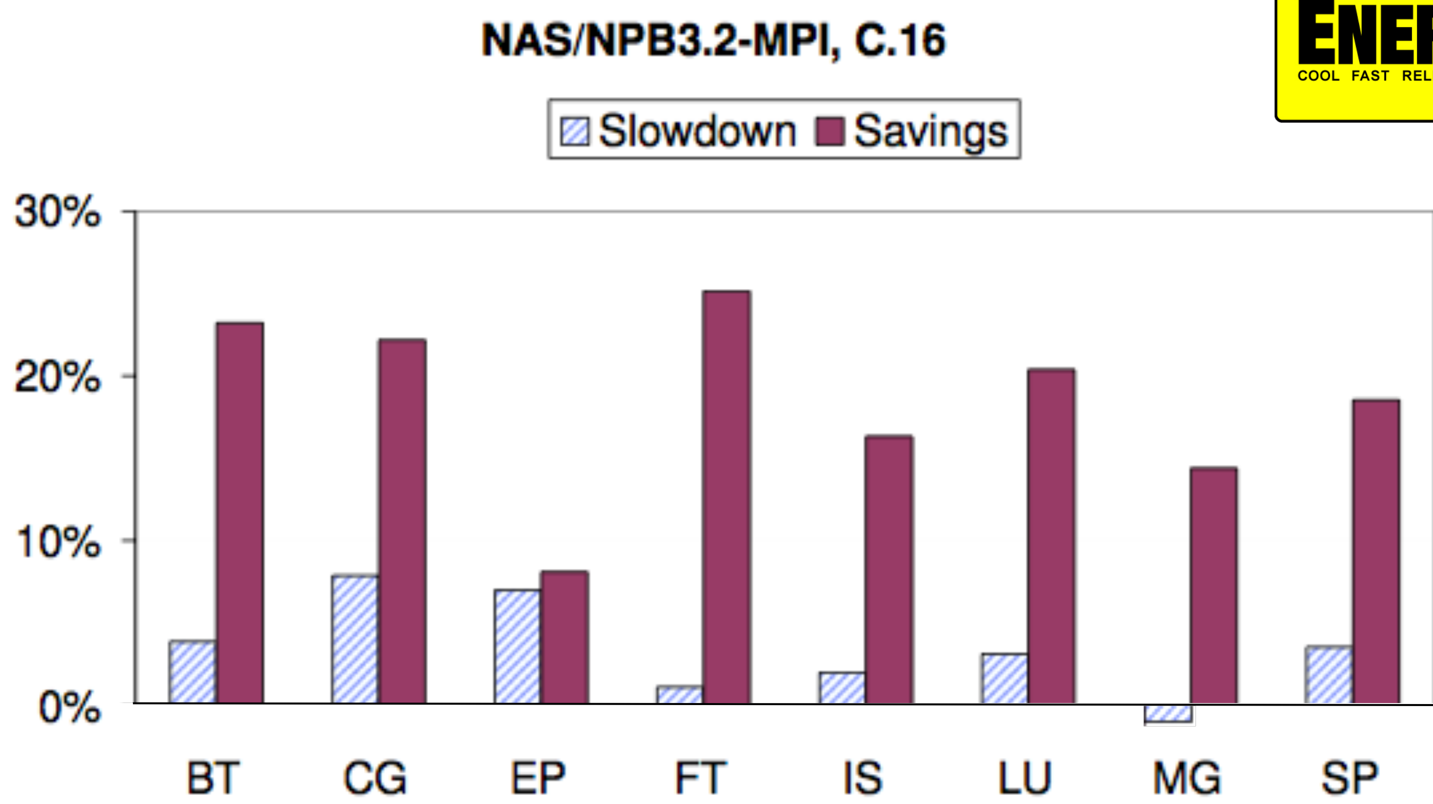
  - The coefficient β is computed at run-time using a regression method on the past MIPS rates reported from the built-in PMU.

$$\beta = \frac{\sum_i (\frac{f_{max}}{f_i} - 1)(\frac{\mathtt{mips}(f_{max})}{\mathtt{mips}(f_i)} - 1)}{\sum_i (\frac{f_{max}}{f_i} - 1)^2}$$

C. Hsu and W. Feng.
"A Power-Aware Run-Time System for High-Performance Computing," *SC|05,* Nov. 2005.

β – Adaptation on NAS Parallel Benchmarks

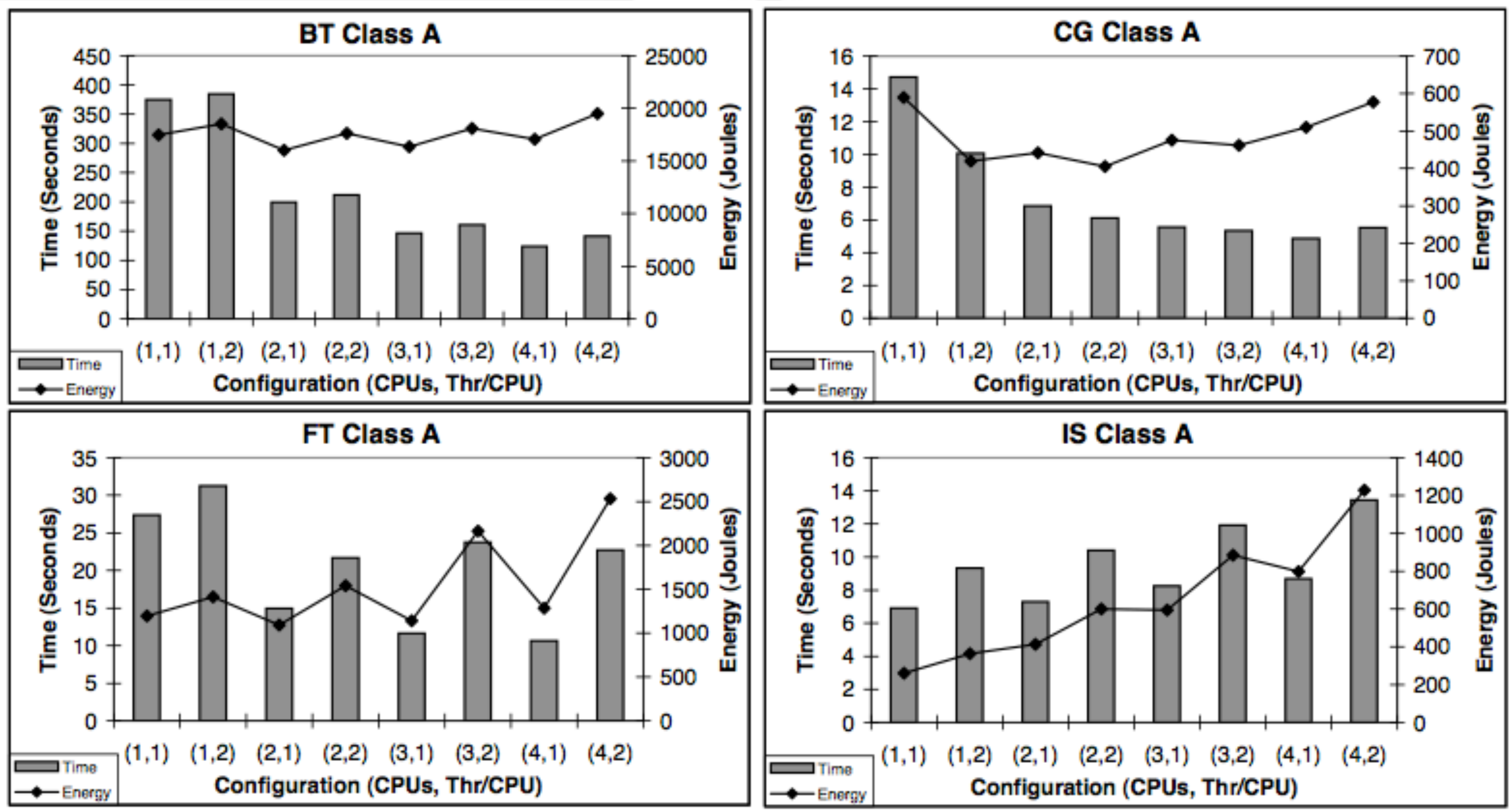
LOS ALAMOS NATIONAL LABORATORY
ENERGYFIT
COOL FAST RELIABLE COMPUTING



Energy reduction (15%)
Performance improvement

C. Hsu and W. Feng.
"A Power-Aware Run-Time System for High-Performance Computing,"
SC|05, Nov. 2005.

Need for Better Performance & Power Modeling

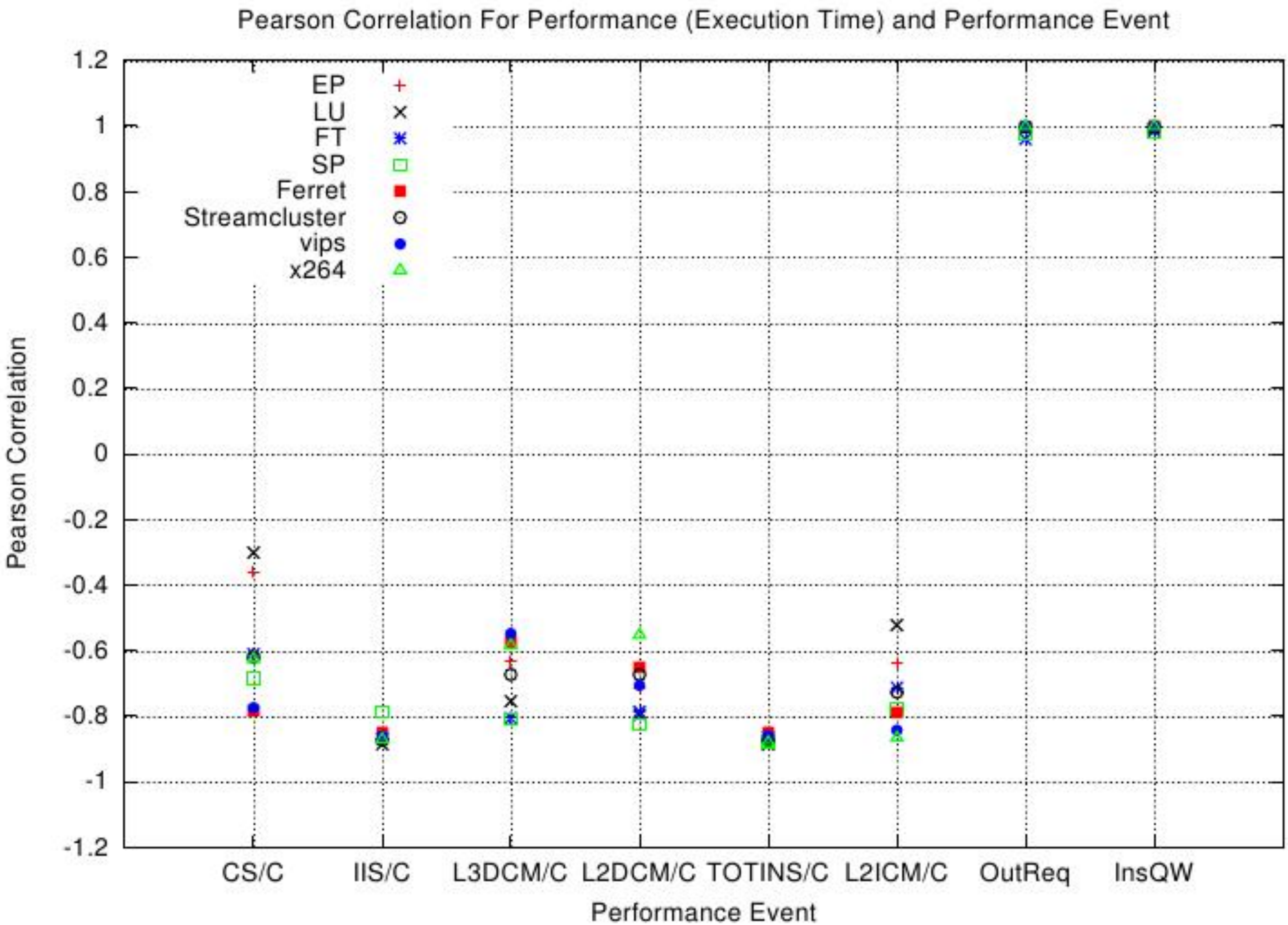


Source: Virginia Tech

Search for Reliable Predictors

- Re-visit performance counters used for prediction
  – Applicability of performance counters across generations of architecture
- Performance counters monitored
  – NPB and PARSEC benchmarks
  – Platform: Intel Xeon E5645 CPU (Westmere-EP)

Context switches (CS)	Instruction issued (IIS)
L3 data cache misses (L3 DCM)	L2 data cache misses (L2 DCM)
L2 instruction cache misses (L2 ICM)	Instruction completed (TOTINS)
Outstanding bus request cycles (OutReq)	Instruction queue write cycles (InsQW)

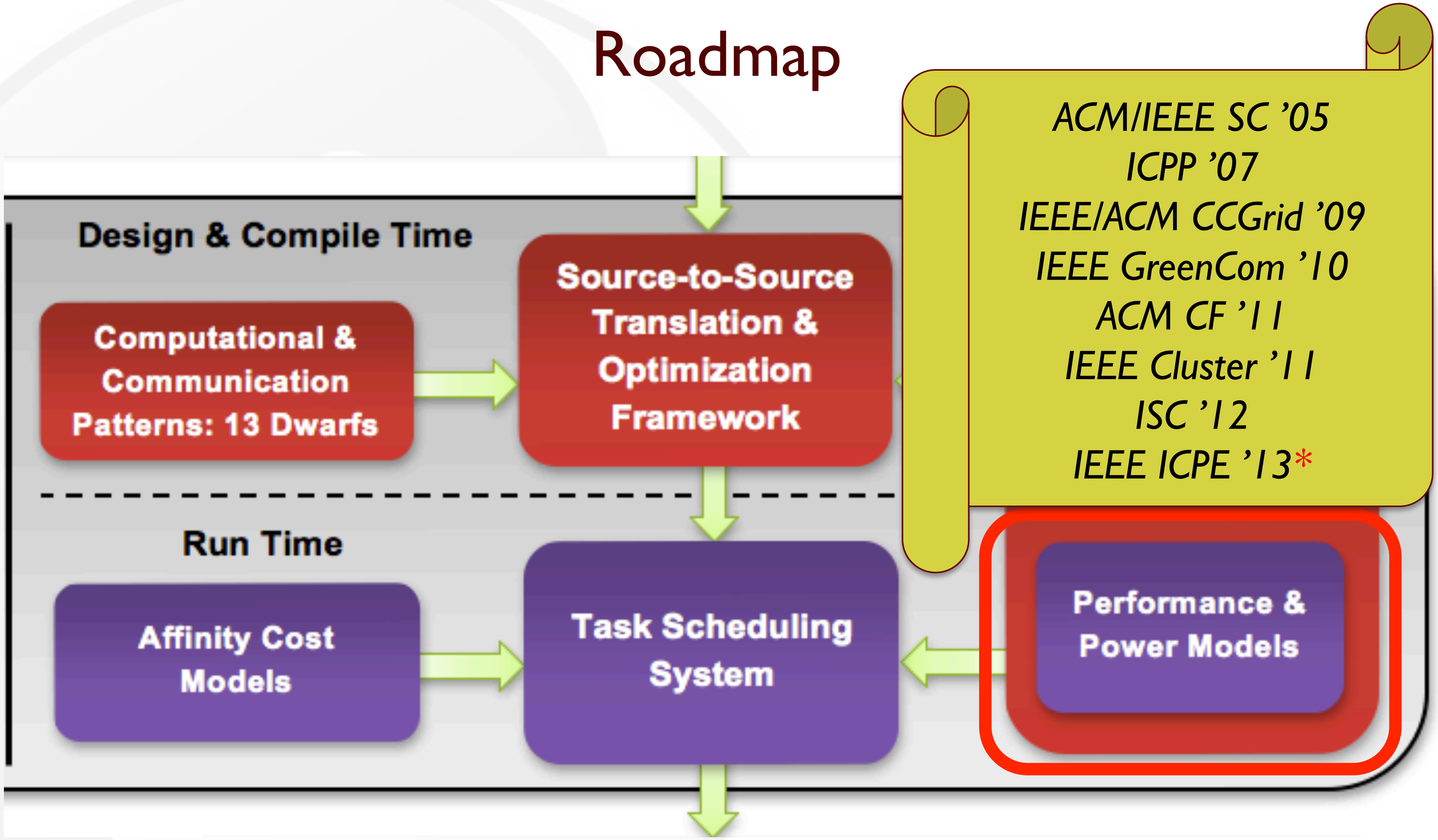


VirginiaTech
Invent the Future

SyNeRG
synergy.cs.vt.edu

Productivity = Performance + Programmability + Portability

Roadmap

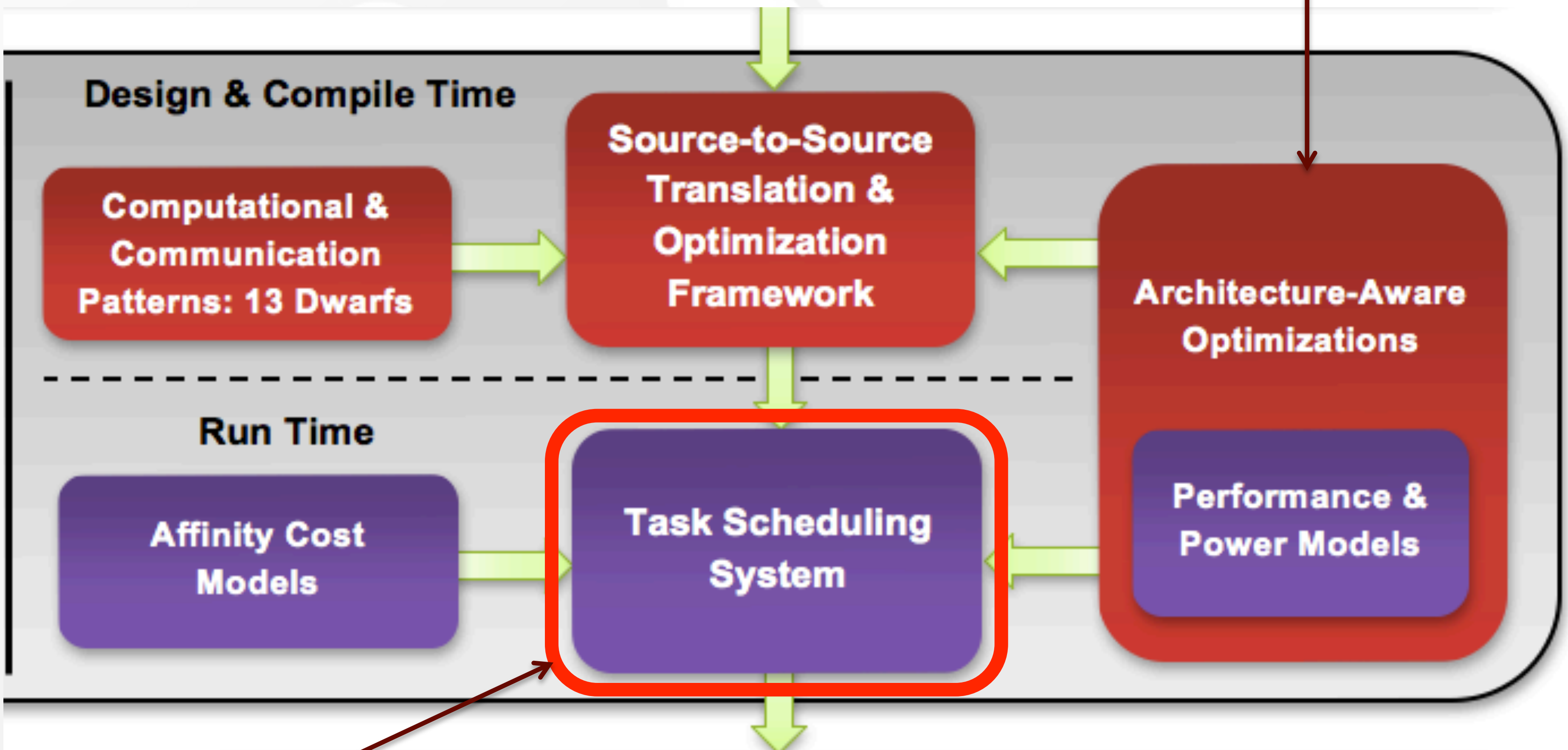


* B. Subramaniam and W. Feng, "Towards Energy-Proportional Computing for Enterprise-Class Server Workloads," 3rd ACM/SPEC Int'l Conf. on Performance Engineering, April 2013. **Best Paper Award**

Productivity = Performance + Programmability + Portability

Roadmap

Performance Portability



Performance Portability

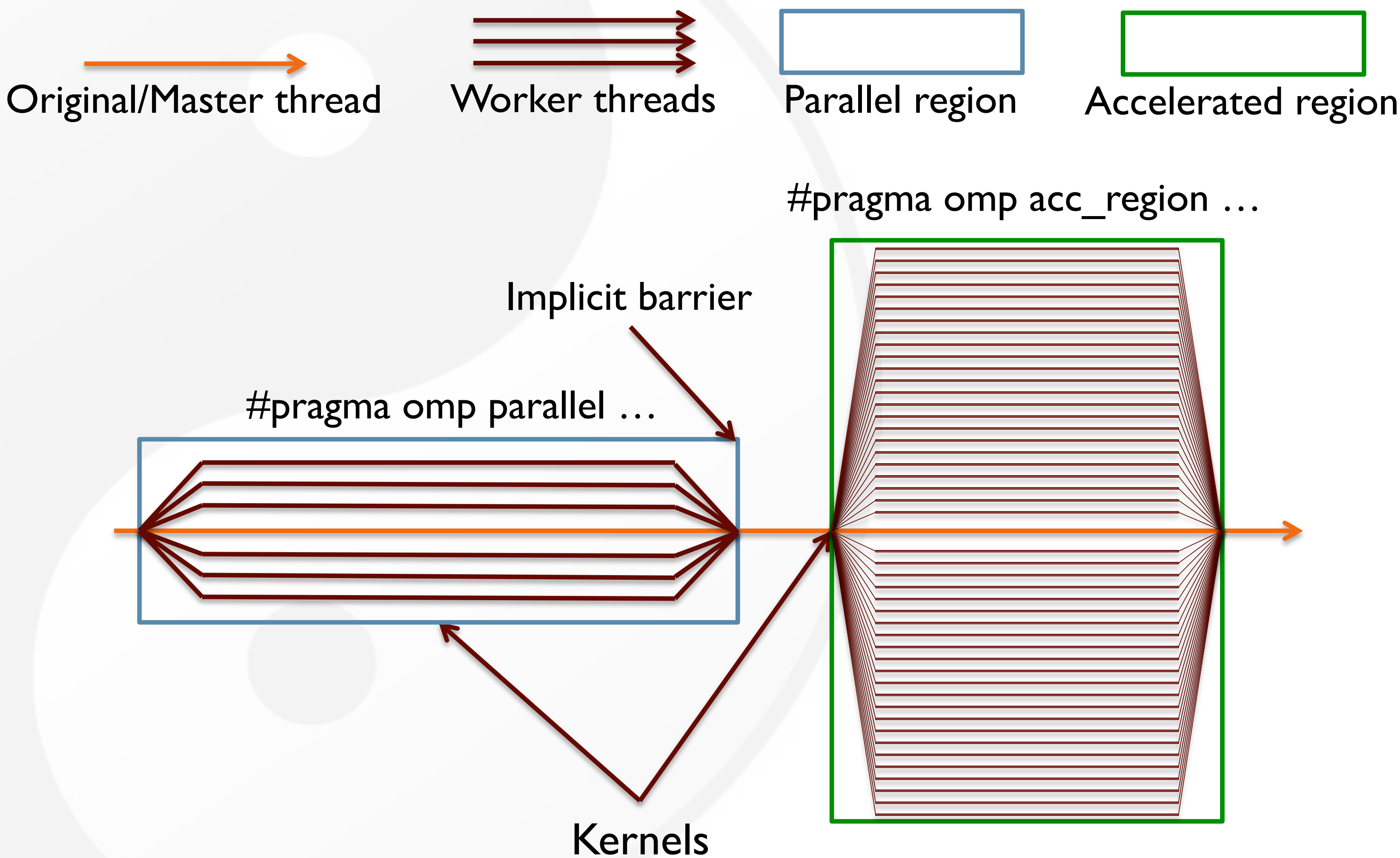
What is Heterogeneous Task Scheduling?

- Automatically spreading tasks across heterogeneous compute resources
  - CPUs, GPUs, APUs, FPGAs, DSPs, and so on
- Specify tasks at a higher level (currently OpenMP extensions)
- Run them across available resources automatically

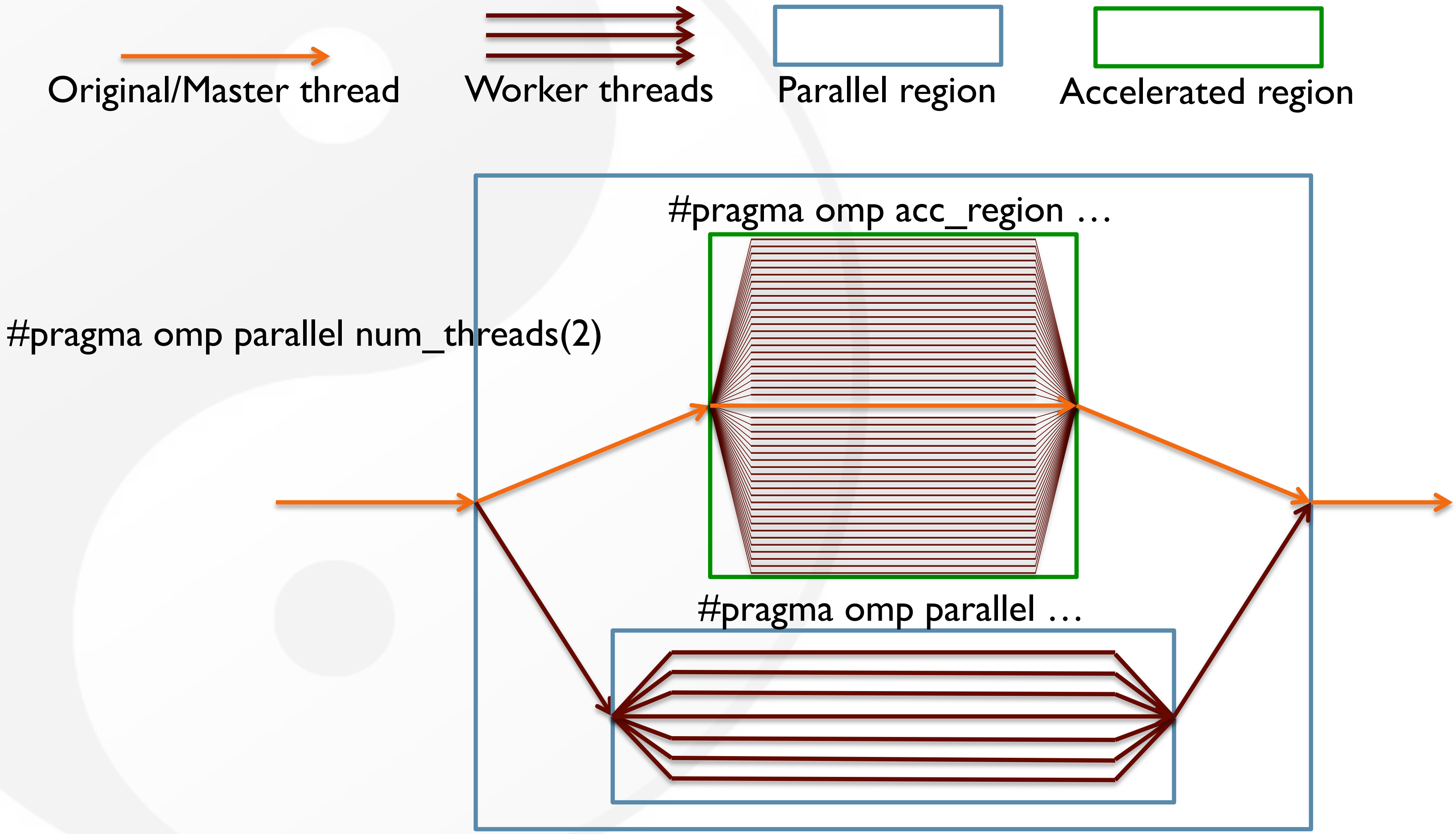
Goal

- A run-time system that intelligently uses what is available resource-wise and optimize for performance portability
  - Each user should *not* have to implement this for themselves!

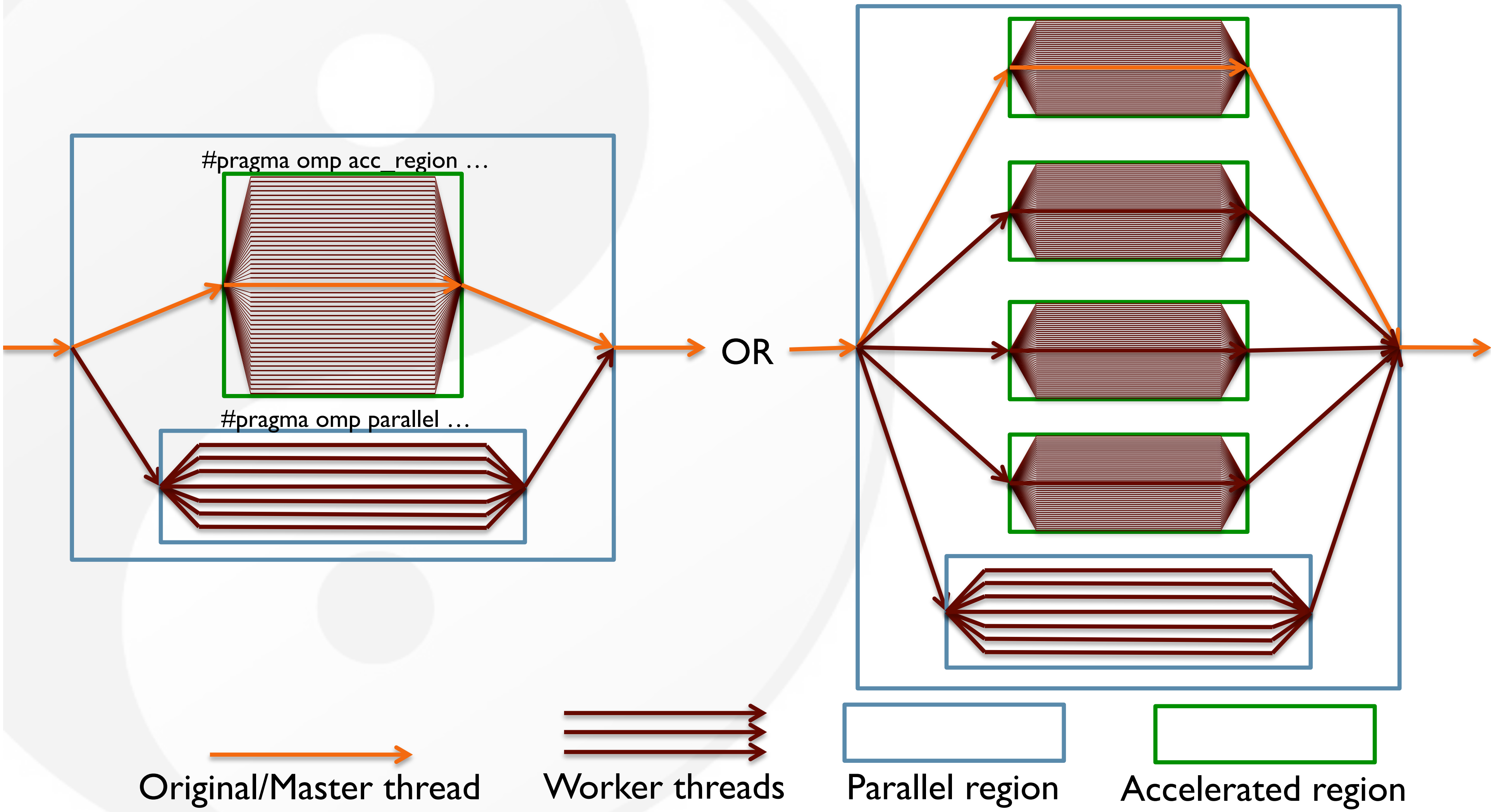
OpenMP Accelerator Behavior



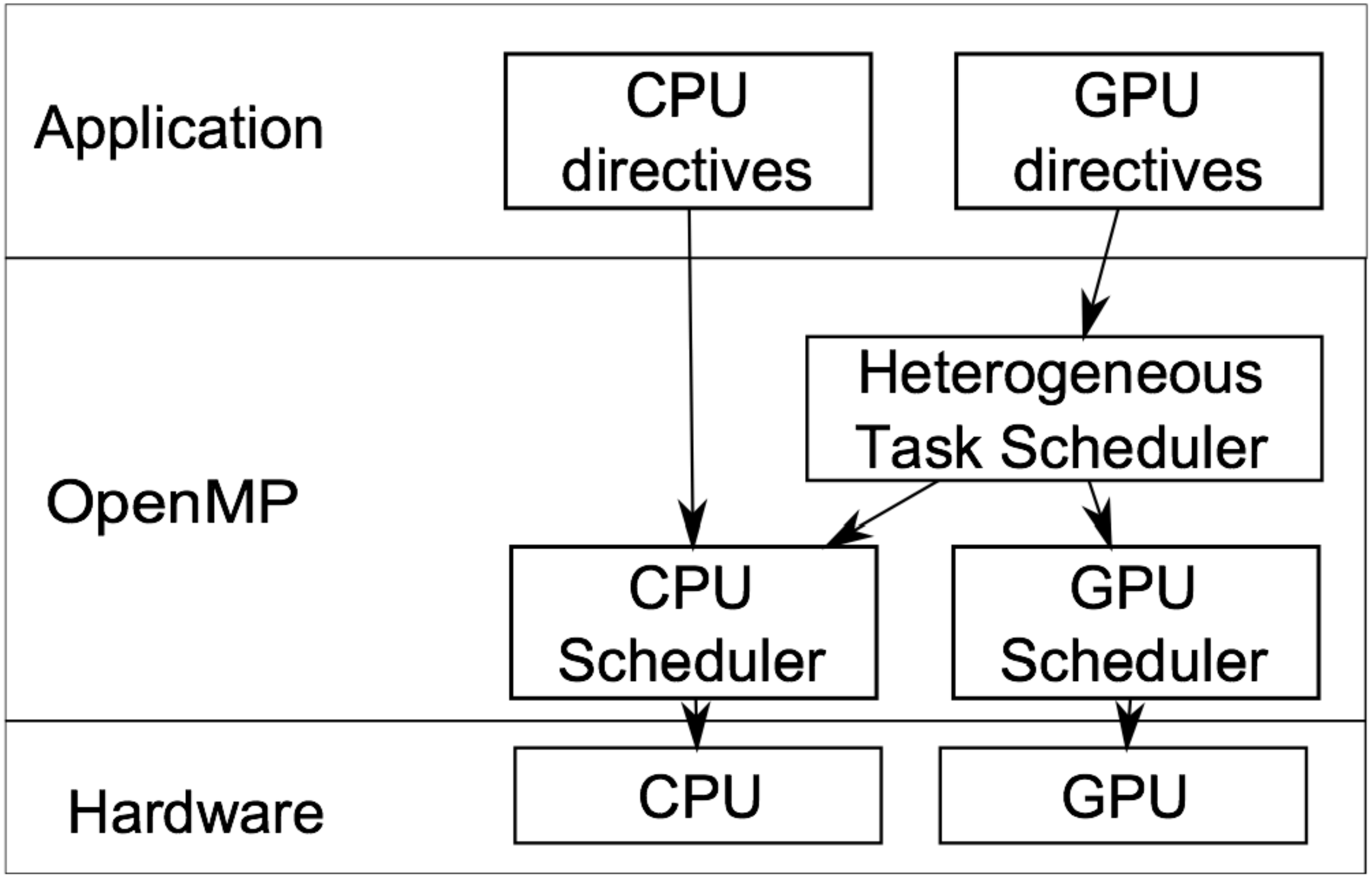
DESIRED OpenMP Accelerator Behavior



Work-share a Region Across the Whole System



Heterogeneous Task Scheduling



How to Heterogeneous Task Schedule (HTS)

- Accelerated OpenMP can offer heterogeneous task scheduling with
  - Programmability
  - Functional portability, given underlying compilers
  - Performance portability
- How?
  - A simple extension to Accelerated OpenMP syntax for **programmability**
  - Automatically dividing parallel tasks across arbitrary heterogeneous compute resources for **functional portability**
    - CPUs
    - GPUs
    - APUs
  - Intelligent runtime task scheduling for **performance portability**

Programmability: Code Transformation

- Manual
  - Add 20 lines
  - Must manually split problem and reassemble results
- Automatic
  - Add 1 clause
  - Splitting and assembly are automatic

```
#pragma omp acc_region_loop private(i)          \
        firstprivate(nco,no,ncl) default(none)  \
        acc_copyin(fc[0:ncl*nco]) present(fo)   \
        acc_copyout(m[0:no])
//      hetero(1,dynamic)
for (i=0; i<no; i++) {
    m[i]  = findc(no,ncl,nco,fo,fc,i);
}
```

Our Proposed Extension

```
splitter * s = split_init(no, SPLIT_DYNAMIC, NULL, NULL);
int *m_c = (int*)malloc(sizeof(int)*no);
for(int d_it=0; d_it < s->d_end; d_it++)
{
    s = split_next(no, d_it);

#pragma omp parallel num_threads(2)
    {
        if(omp_get_thread_num()>0)
        {//CPU OpenMP code
            split_cpu_start(s);
#pragma omp parallel shared(fo,fc,m_c,s)             \
                num_threads(omp_get_thread_limit()-1) \
                firstprivate(no,ncl,nco) private(i)
            {
                #pragma omp for
                for (i=s->cts; i<s->cte; i++) {
                    m_c[i] = findc(no,ncl,nco,fo,fc,i);
                }
            }
            split_cpu_end(s);
        }else{//GPU OpenMP code
            split_gpu_start(s);
            int gts = s->gts, gte = s->gte;
#pragma omp acc_region_loop private(i)               \
                    firstprivate(nco,no,ncl,gts,gte)\
                    acc_copyin(fc[0:ncl*nco])       \
                    acc_copyout(m[0:no])            \
                    present(fo)    default(none)
            for (i=gts; i<gte; i++) {
                m[i] = findc(no,ncl,nco,fo,fc,i);
            }
            split_gpu_end(s);
        }
    }
}
memcpy(m+s->d_ccs,m_c+s->d_ccs,
       (s->d_cce-s->d_ccs)*sizeof(int));
free(m_c);
```

Manual

CoreTSAR: Scheduling and Load-Balancing by Adaptation

- Measure computational suitability at runtime
- Compute new distribution of work through a linear optimization approach
- Re-distribute work before each pass

$$
\begin{aligned}
I &= \text{total iterations available} \\
i_j &= \text{iterations for compute unit j} \\
f_j &= \text{fraction of iterations for compute unit j} \\
p_j &= \text{recent time/iteration for compute unit j} \\
n &= \text{number of compute devices} \\
t_j^+ (\text{or } t_j^-) &= \text{time over (or under) equal}
\end{aligned}
$$

$$min(\sum_{j=1}^{n-1} t_j^+ + t_j^-) \tag{7}$$

$$\sum_{j=1}^{n} f_j = 1 \tag{8}$$

$$f_2 * p_2 - f_1 * p_1 = t_1^+ - t_1^- \tag{9}$$

$$f_3 * p_3 - f_1 * p_1 = t_2^+ - t_2^- \tag{10}$$

$$\vdots$$

$$f_n * p_n - f_1 * p_1 = t_{n-1}^+ - t_{n-1}^- \tag{11}$$

Heterogeneous Scheduling: Issues and Solutions

- Issue: Launching overhead is high on GPUs
  - Using a work queue, many GPU kernels may need to be run
- Solution: Schedule only at the beginning of a region
  - The overhead is only paid once or a small number of times
- Issue: Without a work queue, how do we balance load?
  - The performance of each device must be predicted
- Solution: Allocate different amounts of work
  - For the first pass, predict a reasonable initial division of work. We use a ratio between the number of CPU and GPU cores for this.
  - For subsequent passes, use the performance of previous passes to predict following passes.

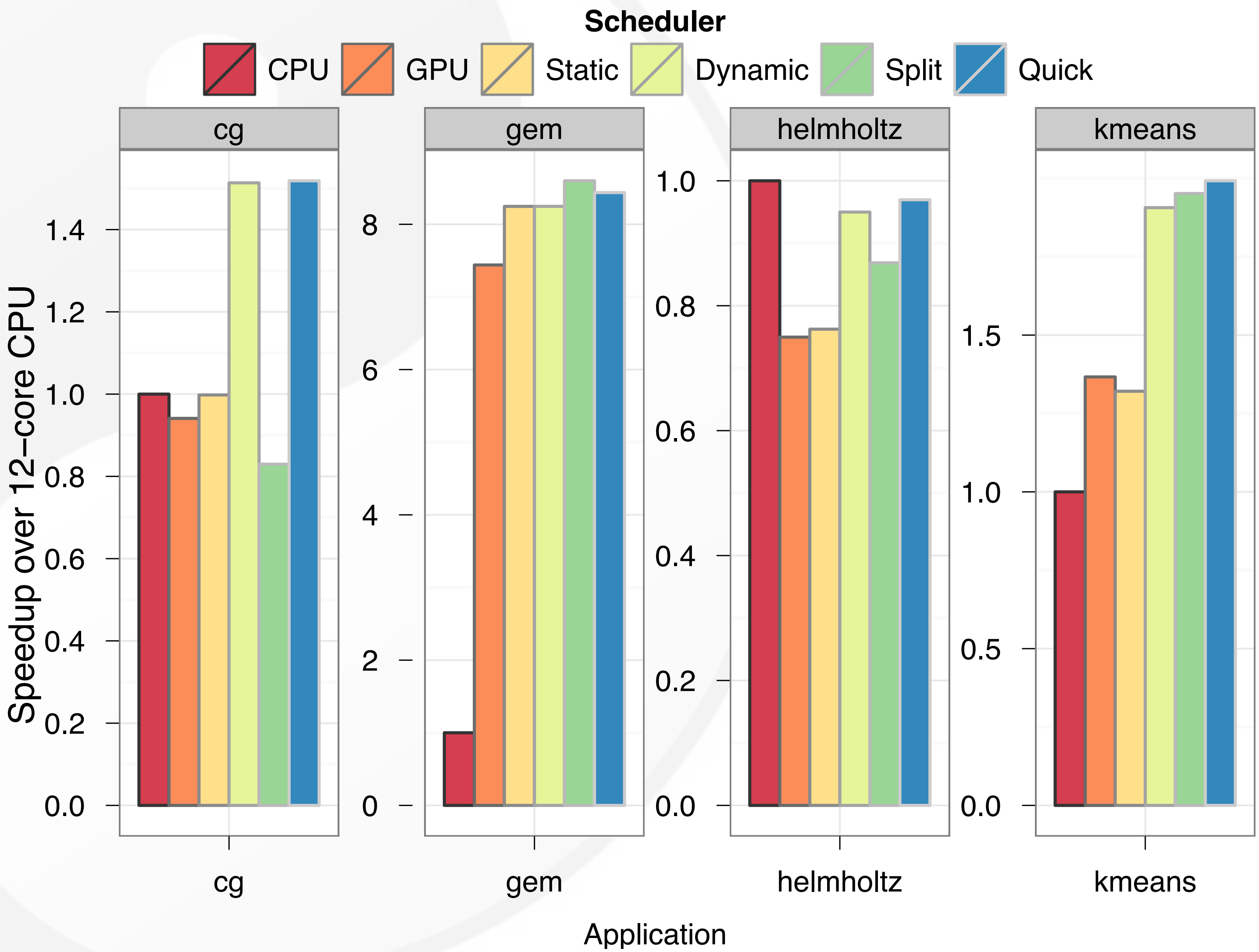
Benchmarks

- NAS CG – Many passes (1,800 for C class)
  - The Conjugate Gradient benchmark from the NAS Parallel Benchmarks
- GEM – One pass
  - Molecular Modeling, computes the electrostatic potential along the surface of a macromolecule
- K-Means – Few passes
  - Iterative clustering of points
- Helmholtz – Few passes, GPU unsuitable
  - Jacobi iterative method implementing the Helmholtz equation

Experimental Setup

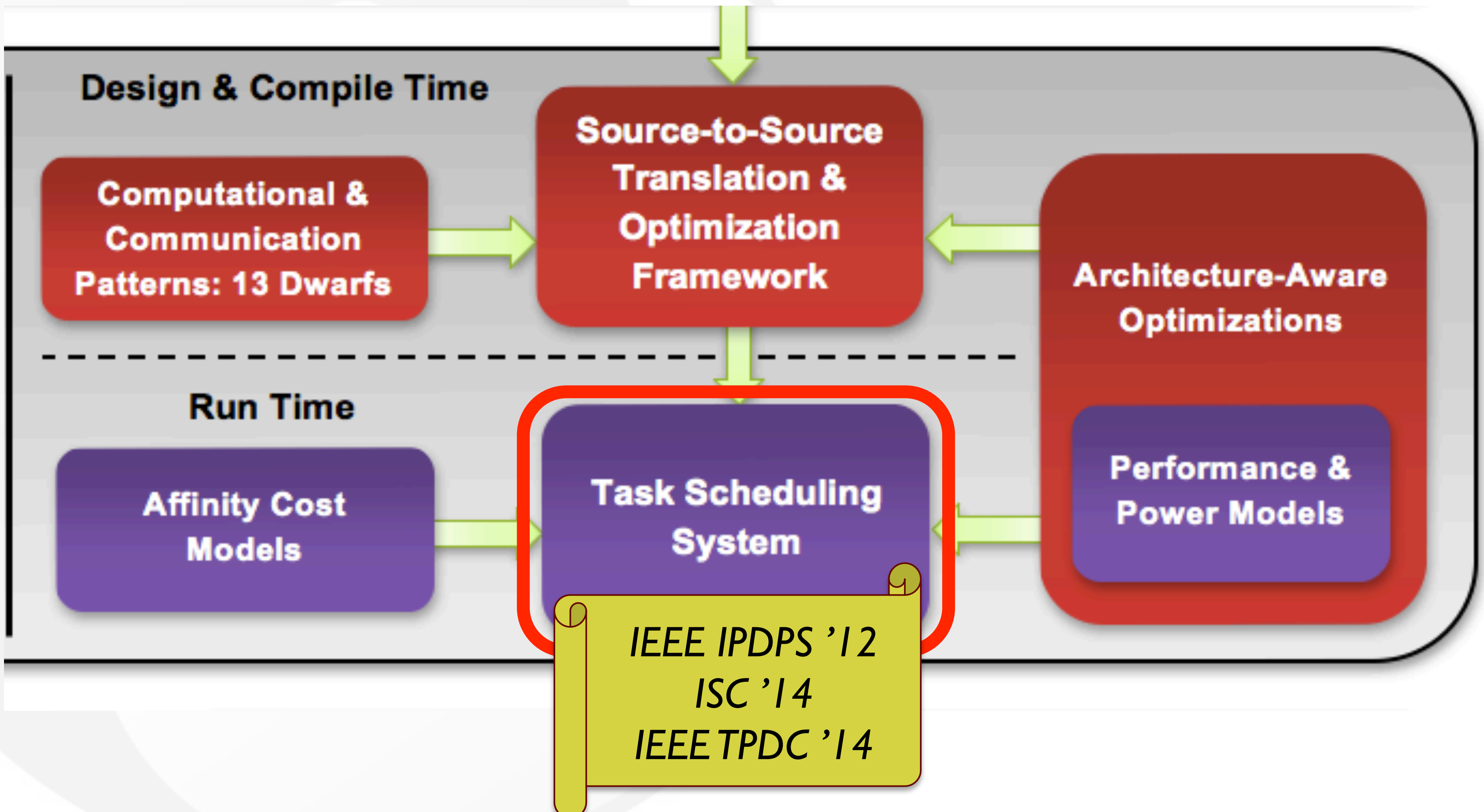
- System
  - 12-core AMD Opteron 6174 CPU
  - NVIDIA Tesla C2050 GPU
  - Linux 2.6.27.19
  - CCE compiler with OpenMP accelerator extensions
- Procedures
  - All parameters default, unless otherwise specified
  - Results represent five or more runs

CoreTSAR Results Across Schedulers

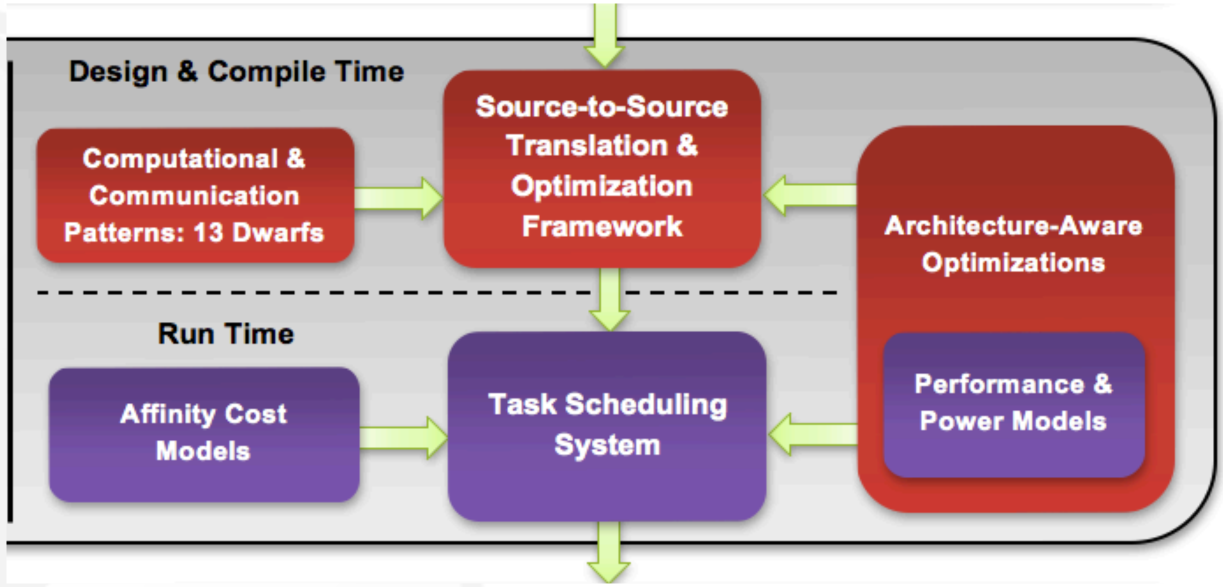


Productivity = Performance + Programmability + Portability

Roadmap



Intra-Node
An Ecosystem for ^ Heterogeneous Parallel Computing



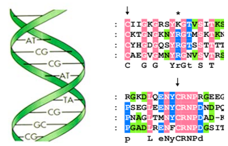
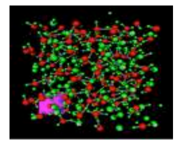
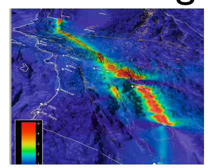
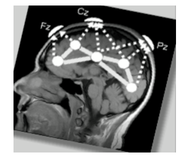
Much work still to be done.

- OpenCL and the 13 Dwarfs — Beta release pending
- Source-to-Source Translation — CU2CL only & no optimization
- Architecture-Aware Optimization — Only manual optimizations
- Performance & Power Modeling — Preliminary & pre-multicore
- Affinity-Based Cost Modeling — Empirical results; modeling in progress
- Heterogeneous Task Scheduling — Preliminary with OpenMP

Intra-Node

An ^ Ecosystem for Heterogeneous Parallel Computing

Applications

Sequence Alignment	Molecular Dynamics	Earthquake Modeling	Neuro-informatics	Avionic Composites
				

Software Ecosystem

Design & Compile Time

Computational & Communication Patterns: 13

Source-to-Source Translation & Optimization

cture-Aware mizations

Run

Affinity ost Models

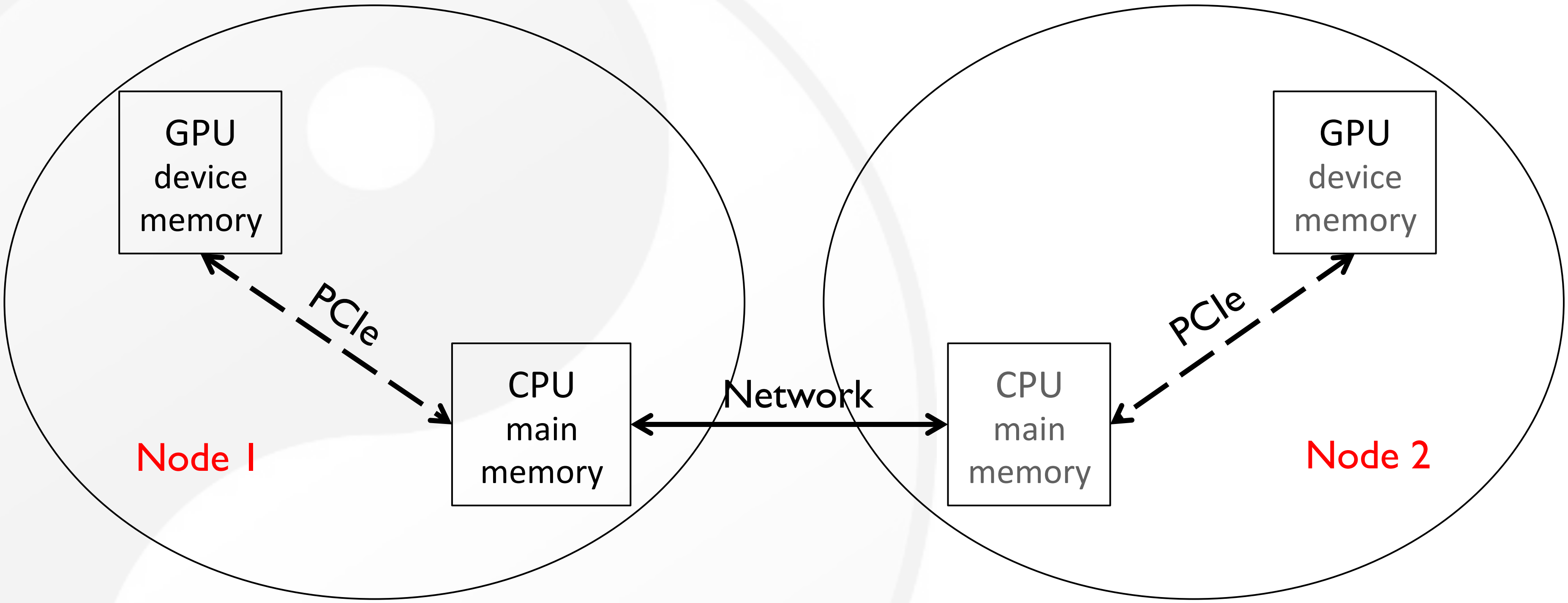
Task Scheduling System

rmance & Power Models

… from intra-node … to inter-node …

Heterogeneous Parallel Computing (HPC) Platform

Programming CPU-GPU Clusters (e.g., MPI+CUDA)



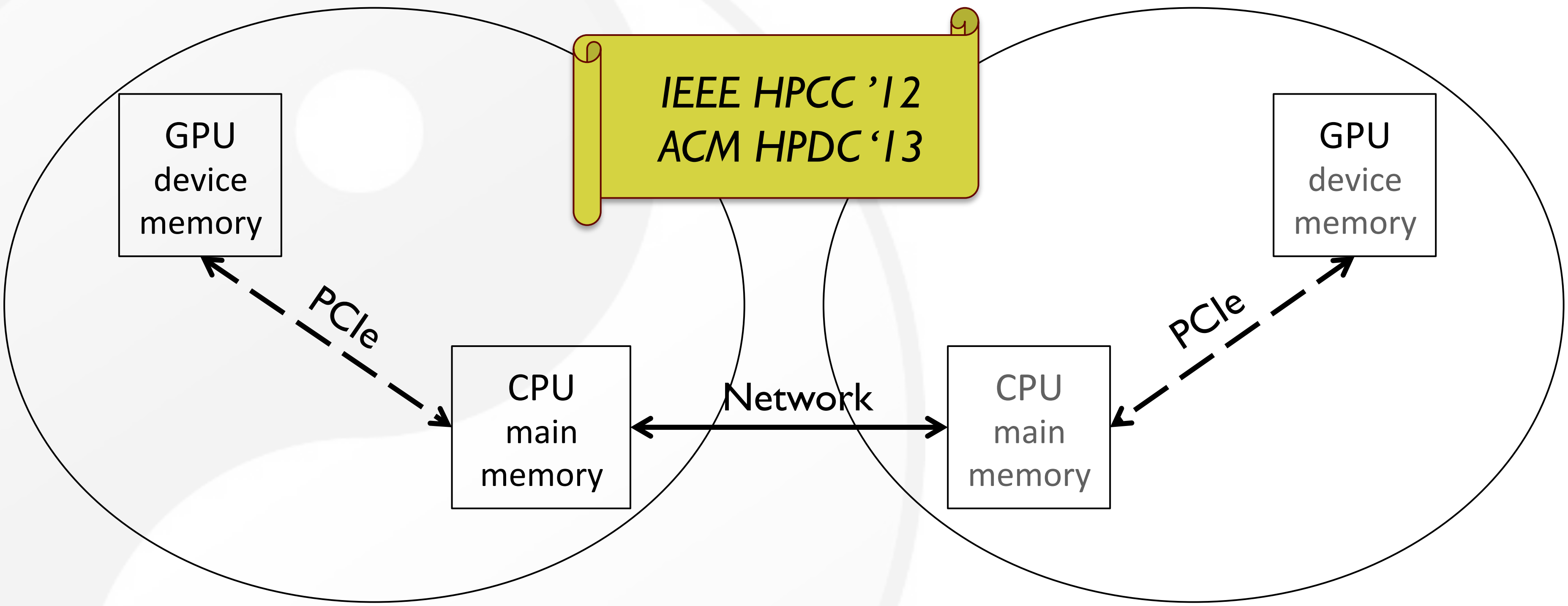
Rank = 0

Rank = 1

```
if(rank == 0)
{
  cudaMemcpy(host_buf, dev_buf, D2H)
  MPI_Send(host_buf, .. ..)
}
```

```
if(rank == 1)
{
  MPI_Recv(host_buf, .. ..)
  cudaMemcpy(dev_buf, host_buf, H2D)
}
```

Goal of Programming CPU-GPU Clusters (MPI + Any Acc)



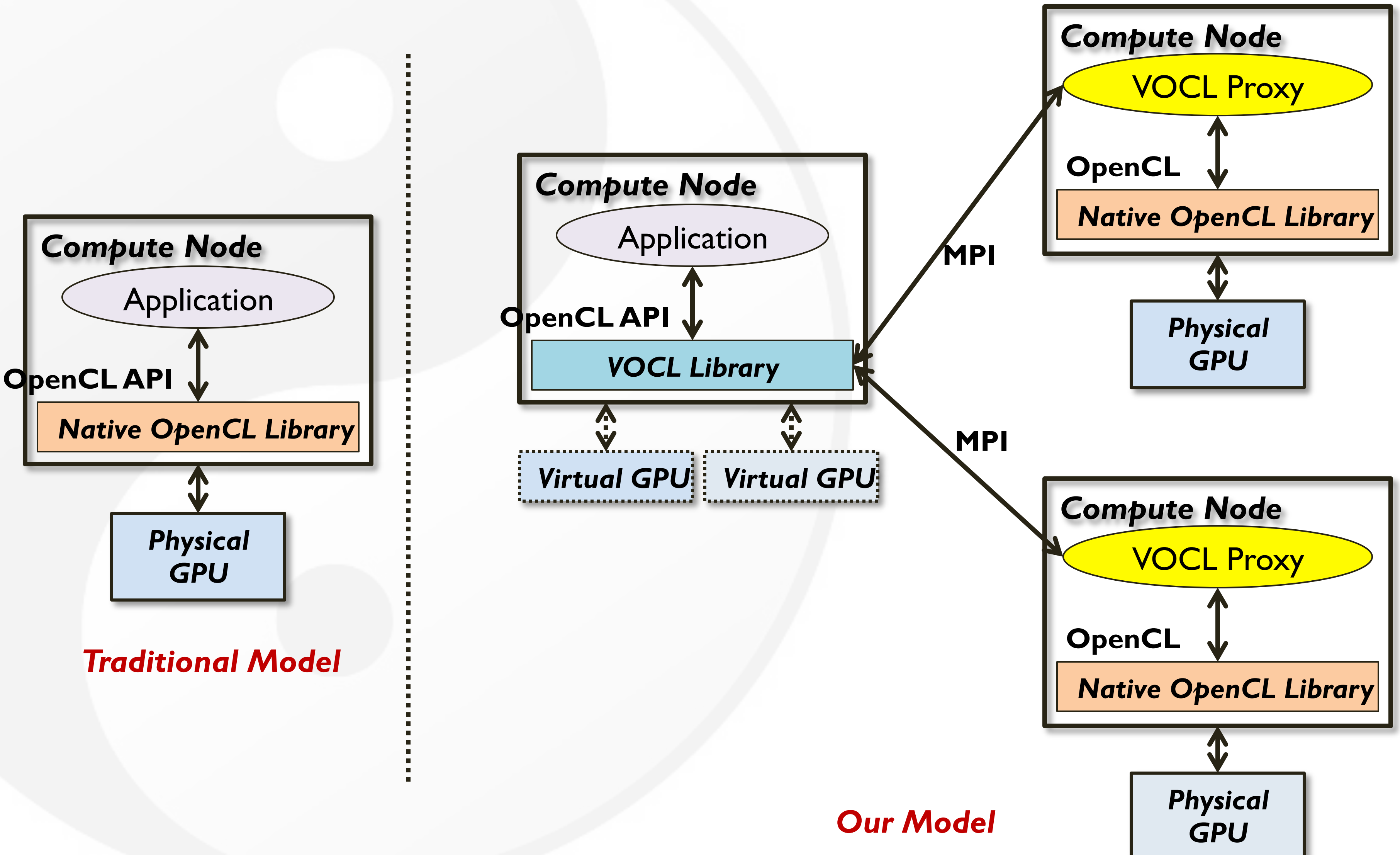
Rank = 0

```
if(rank == 0)
{
  MPI_Send(any_buf, .. ..);
}
```

Rank = 1

```
if(rank == 1)
{
  MPI_Recv(any_buf, .. ..);
}
```

"Virtualizing" GPUs …

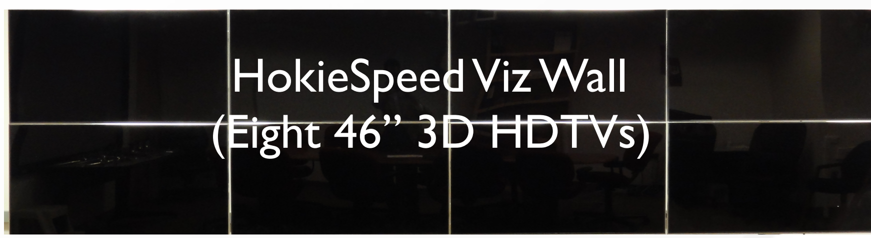


Heterogeneous HPC @VT

HOKIESPEED

- CPU+GPU (i.e., “left brain”+”right brain”) heterogeneous supercomputer *with large-scale visualization wall*
- Peak Performance
  - 455 trillion floating-point operations / second
- Debuted as the ***GREENEST*** commodity supercomputer in the U.S. in Nov. 2011

THE GREEN 500



- Cost: ~ $1M (vs. $1250M for *K Computer*)

SyNeRGy Lab @ Virginia Tech

HOME | NEWS | RESEARCH | PUBLICATIONS | PEOPLE | VIDEOS | PICTURES | WIKI

SEEC · CU2CL · CHREC · Green500 · HokieSpeed · mpiBLAST · MyVICE · SSS

Synergy People

Address: Torgersen Hall 2050; 620 Drillfield Drive (Alumni Mall); Blacksburg, VA 24061

Faculty & Staff

Wu-chun ("Wu") Feng
Dept. of CS, ECE, VBI at Virginia Tech and Dept. of Cancer Biology & Translational Science Inst. at Wake Forest U.

Mark K. Gardner
Office of IT and Affiliate Faculty Dept. of CS.

Paul Sathre
Postmasters Research Associate, Dept. of CS.

Missy Thomas
Administrator for Synergy Lab and SEEC Center.

Nataliya E. Timoshevskaya
Postdoctoral Research Associate, Virginia Tech.

Harold Trease
Sr. Research Scientist, Dept. of CS.

Hao Wang
Postdoctoral Research Associate, Dept. of CS.

Students

Ashwin Aji
(Ph.D.)
NVIDIA Graduate Fellowship

Vignesh Adhinarayanan
(Ph.D.)

Xuewen Cui
(Ph.D.)

L. R. Sriram Chivukula
(M.S.)

Sajal Dash
(Ph.D.)

Islam Harb
(Ph.D.)

Ahmed Helal
(Ph.D.)

Kaixi Hou
(Ph.D.)

Rubasri Kalidas
(M.S.)

Umar Kalim
(Ph.D.)

Konstantinos Krommydas
(Ph.D.)

Sarunya (Kwang) Pumma
(Ph.D.)

Thomas Scogland
(Ph.D.)
NDSEG Fellowship

Balaji Subramaniam
(Ph.D.)

Xiaodong Yu
(Ph.D.)

Da Zhang
(Ph.D.)

Jing Zhang
(Ph.D.)

Collaborators (Current & Past)

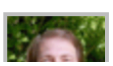
Peter Athanas
Dept. of ECE, Virginia Tech.

Pavan Balaji
Argonne National Laboratory.

Keith Bisset
Virginia Bioinformatics Institute, Virginia Tech.

Eric Brown
Office of IT, Virginia Tech.

Yong Cao
Dept. of CS, Virginia Tech.

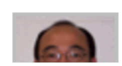


Bronis de Supinski
Lawrence Livermore

Jack Edwards
Dept. of MAE,

Xiaohui (Helen) Gu
Dept. of CS,

Chung-Hsing Hsu
Oak Ridge National



Hong Luo
Dept. of MAE,

Funding Acknowledgements

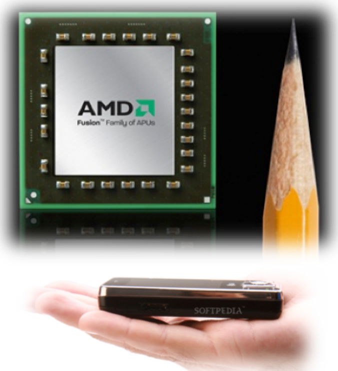
Microsoft featured our "Big Data in the Cloud" grant in U.S. Congressional Testimony

Conclusion

- An ecosystem for heterogeneous parallel computing

#96 on TOP500 (11/11)

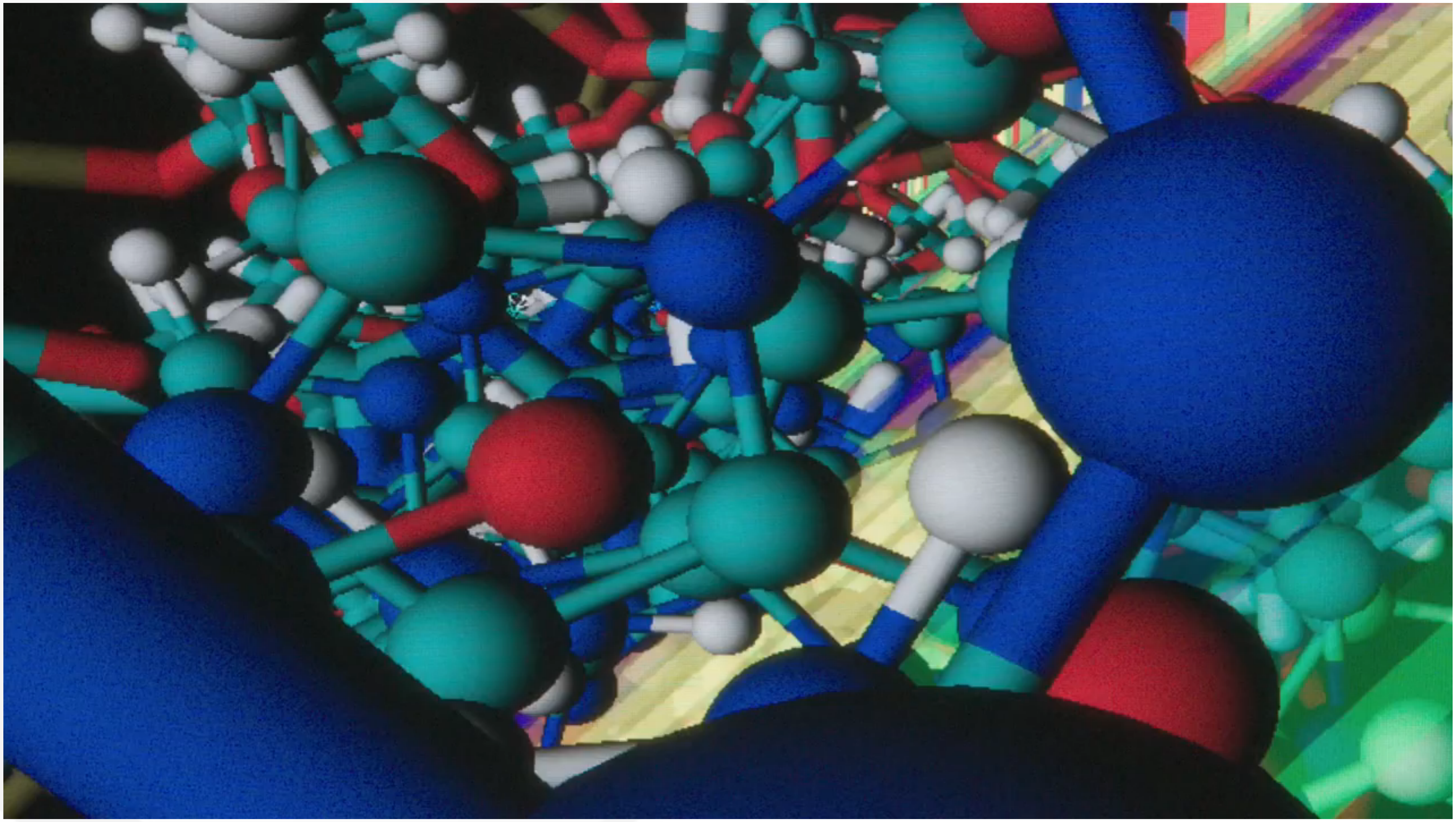
Highest-ranked commodity supercomputer in U.S. on the Green500 (11/11)



  - Enabling software that tunes parameters of hardware devices
    … *with respect to* ***performance, programmability, and portability***
    … *via a benchmark suite of dwarfs (i.e., motifs) and mini-apps*

Microsoft Cloud Commercial:

Data-Intensive Biocomputing in the Cloud



Virginia Tech
Invent the Future

Los Alamos National Laboratory

SyNeRGy
synergy.cs.vt.edu

Microsoft Cloud Infomercial:

Data-Intensive Biocomputing in the Cloud

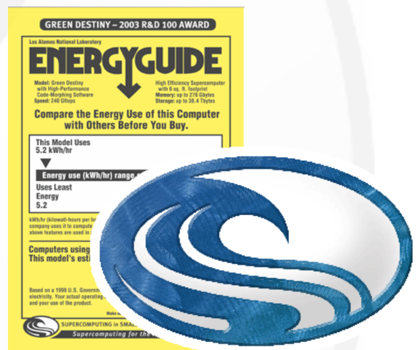
VirginiaTech
Invent the Future

SyNeRG
synergy.cs.vt.edu

Wu Feng, wfeng@vt.edu, 540-231-1192

http://synergy.cs.vt.edu/

http://www.chrec.org/

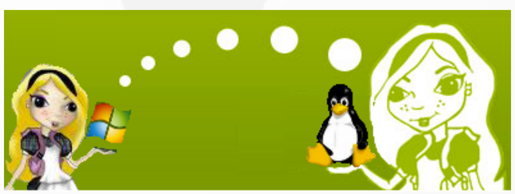


http://www.mpiblast.org/

SUPERCOMPUTING in SMALL SPACES

http://sss.cs.vt.edu/

http://www.green500.org/



http://myvice.cs.vt.edu/

"Accelerators 'R Us"

http://accel.cs.vt.edu/